5 System Maintenance

5.1 System Overview

5.1.1 Class Descriptions

- **RefractionSimulator**: a window of the application which starts the rest of the application.
- **UIController**: contains methods for generating and regenerating parts of the user interface and returning them so that the window can display them.
    - **BeamThicknessActionListener**: validates the information entered into a beam thickness text field, rounds it to an integer between 1 and 10 inclusive or the last valid entry and updates the geometry of the beam object.
    - **IncDecActionListener**: enters an appropriate new beam thickness when the user clicks one of the buttons to increment or decrement the beam thickness.
    - **MaterialNameActionListener**: trims whitespace from the beginning and end of the string entered into a custom material name field, restricts the result to 30 characters (and trims again) and replaces a blank name with "Custom material".
    - **RefractiveIndexActionListener**: validates the information entered into a new material refractive index field and restricts it to a real number between 1 and 100 inclusive.
    - **SliderListener**: when a slider is dragged, this updates other sliders and the position and orientation of the selected ray box.
    - **TextFieldFocusListener**: triggers a text field's action listener when it loses focus (typically when the user clicks on something else) so that the input can be validated and altered appropriately.
    - **FileSaver**: allows dialog boxes to be created which let the user save an image to a secondary storage device.
- **Viewport**: a user interface component which displays images of 3-D space in real-time as well as handling all of the 3-D objects and updating other areas of the user interface when called to do so by the ViewportListener object.
- **ViewportListener**: responds to both mouse and keyboard events that involve its Viewport object and often call the Viewport object's methods in order to update 3-D space or the user interface.
- **Object3D**: allows for the creation of generic three-dimensional objects and provides methods for manipulating these objects
- **Target**: 3-D objects which are more specifically target objects, meaning they have a material and must be one of the primitive shapes.
- **RayBox**: 3-D objects which are more specifically ray boxes, meaning they are cubes with centre five units from the world's origin and each has its own Beam object.
- **Beam**: 3-D objects which represent beams of light and as such their geometry is dependent on their origin's position, their orientation, the shape of the target object and the materials of the world and target object.
- **Mesh**: represents the geometry (vertices and triangular faces) of a 3-D object as well as its arbitrarily orientated bounding box.

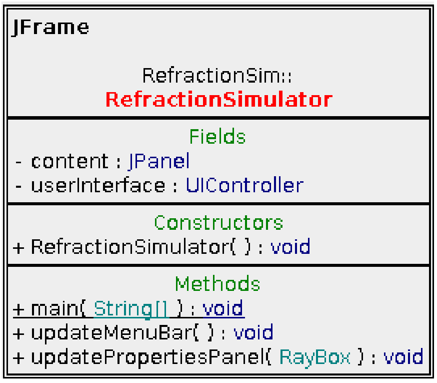
    - **Primitive:** an enumerated type that specifies the preset shapes for which the application can generate geometry.
- **Matrix**: represent transformations in n-dimensional space such as projections, rotations and enlargements.
- **Vector**: represent positions or displacements in n-dimensional space for operations that involve matrices.
- **EulerTriple**: represent orientations and angular displacements in 3-D space using Euler angle triples (heading, pitch and bank). Methods are provided for adding angular displacements and converting the EulerTriple to a Matrix object.
- **Ray**: represent lines in three dimensions each with a starting point and a direction. These are used in calculating the path of a beam of light**.**
- **Edge2D**: represent edges (line segments) in two dimensions. These are used in rendering faces and detecting if a ray passes through a particular face.
- **Math2**: contains static methods (accessible without creating an instance of the class) which perform general-purpose mathematical operations not already provided by the Math class.

5.1.2 Class Properties and Methods

Key:

+ public

- private

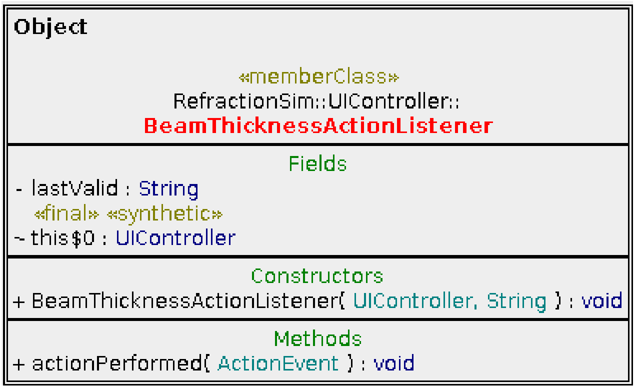

JFrame

RefractionSim::
RefractionSimulator

Fields
- content : JPanel
- userInterface : UIController

Constructors
+ RefractionSimulator( ) : void

Methods
+ main(String[]) : void
+ updateMenuBar( ) : void
+ updatePropertiesPanel(RayBox) : void

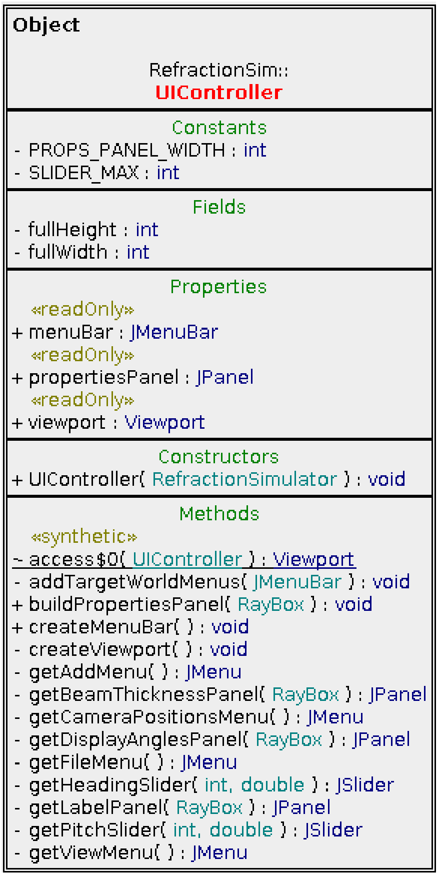

Object

«memberClass»
RefractionSim::UIController::
BeamThicknessActionListener

Fields
- lastValid : String
«final» «synthetic»
~ this$0 : UIController

Constructors
+ BeamThicknessActionListener(UIController, String) : void

Methods
+ actionPerformed(ActionEvent) : void


Object

RefractionSim::
UIController

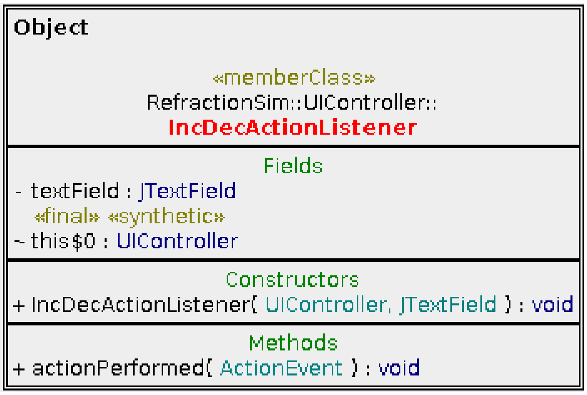
Constants
- PROPS_PANEL_WIDTH : int
- SLIDER_MAX : int

Fields
- fullHeight : int
- fullWidth : int

Properties
«readOnly»
+ menuBar : JMenuBar
«readOnly»
+ propertiesPanel : JPanel
«readOnly»
+ viewport : Viewport

Constructors
+ UIController(RefractionSimulator) : void

Methods
«synthetic»
~ access$0(UIController) : Viewport
- addTargetWorldMenus(JMenuBar) : void
+ buildPropertiesPanel(RayBox) : void
+ createMenuBar( ) : void
- createViewport( ) : void
- getAddMenu( ) : JMenu
- getBeamThicknessPanel(RayBox) : JPanel
- getCameraPositionsMenu( ) : JMenu
- getDisplayAnglesPanel(RayBox) : JPanel
- getFileMenu( ) : JMenu
- getHeadingSlider(int, double) : JSlider
- getLabelPanel(RayBox) : JPanel
- getPitchSlider(int, double) : JSlider
- getViewMenu( ) : JMenu

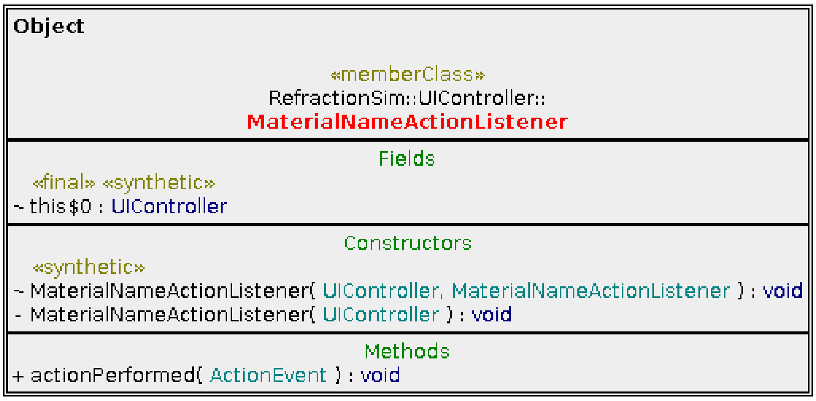

Object

«memberClass»
RefractionSim::UIController::
IncDecActionListener

Fields
- textField : JTextField
«final» «synthetic»
~ this$0 : UIController

Constructors
+ IncDecActionListener(UIController, JTextField) : void

Methods
+ actionPerformed(ActionEvent) : void

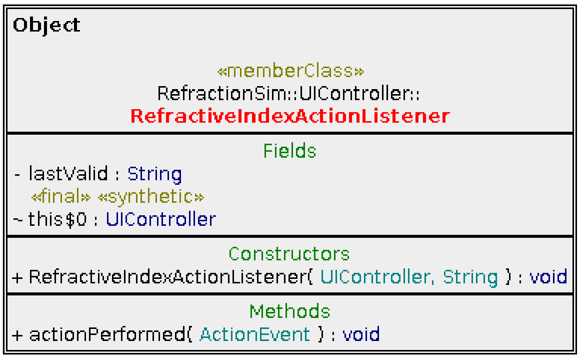

Object

«memberClass»
RefractionSim::UIController::
MaterialNameActionListener

Fields
«final» «synthetic»
~ this$0 : UIController

Constructors
«synthetic»
~ MaterialNameActionListener(UIController, MaterialNameActionListener) : void
- MaterialNameActionListener(UIController) : void

Methods
+ actionPerformed(ActionEvent) : void

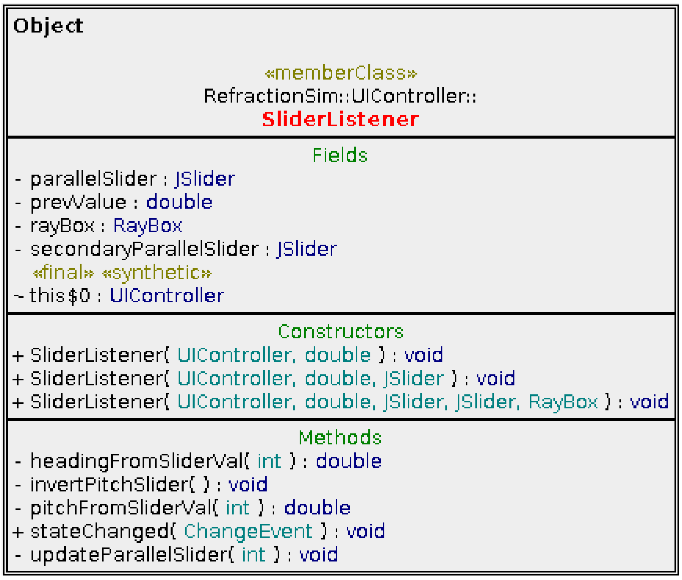

Object

«memberClass»
RefractionSim::UIController::
RefractiveIndexActionListener

Fields
- lastValid : String
«final» «synthetic»
~ this$0 : UIController

Constructors
+ RefractiveIndexActionListener(UIController, String) : void

Methods
+ actionPerformed(ActionEvent) : void

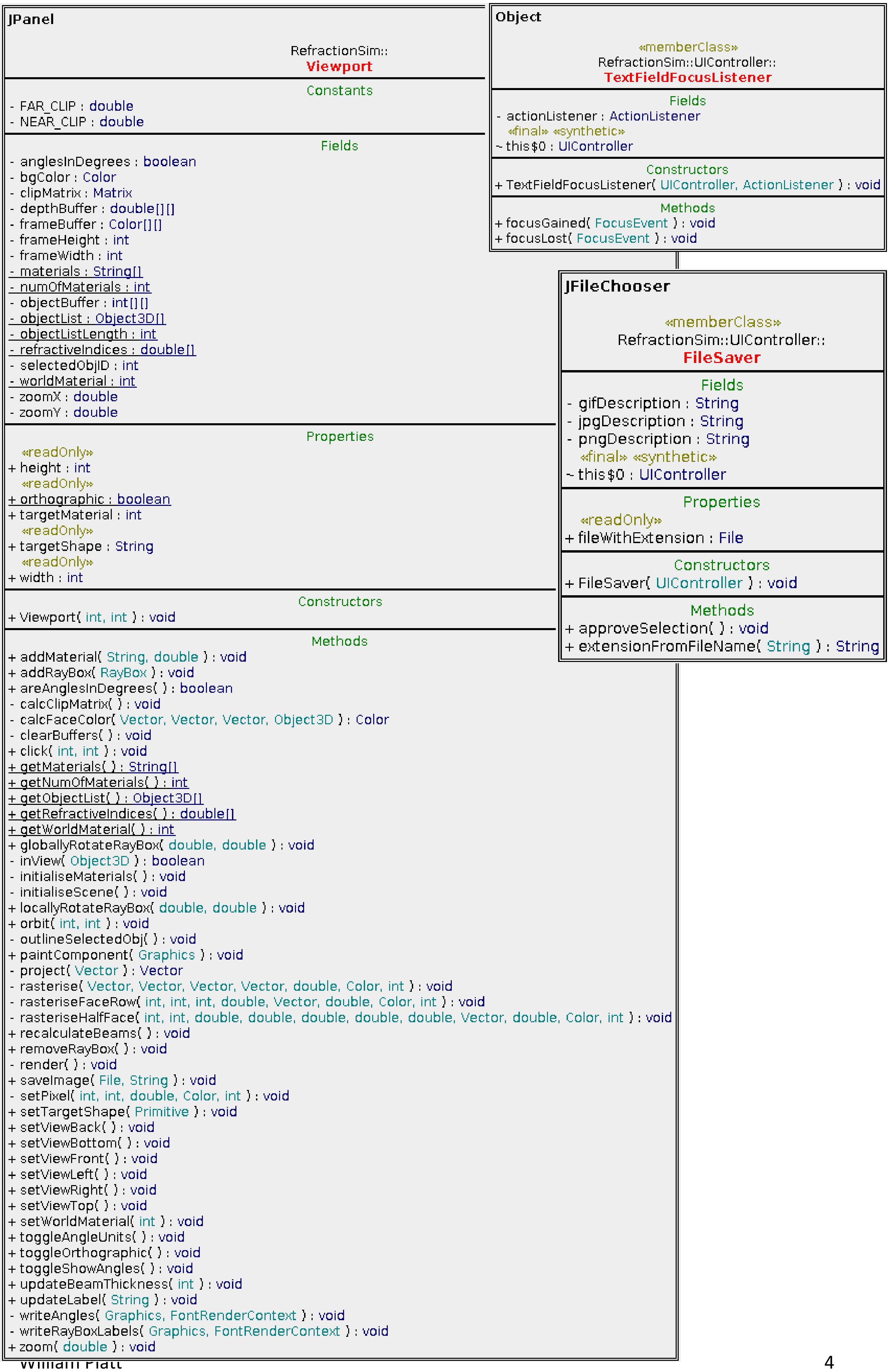

Object

«memberClass»
RefractionSim::UIController::
SliderListener

Fields
- parallelSlider : JSlider
- prevValue : double
- rayBox : RayBox
- secondaryParallelSlider : JSlider
«final» «synthetic»
~ this$0 : UIController

Constructors
+ SliderListener(UIController, double) : void
+ SliderListener(UIController, double, JSlider) : void
+ SliderListener(UIController, double, JSlider, JSlider, RayBox) : void

Methods
- headingFromSliderVal(int) : double
- invertPitchSlider( ) : void
- pitchFromSliderVal(int) : double
+ stateChanged(ChangeEvent) : void
- updateParallelSlider(int) : void



JPanel

RefractionSim::
Viewport

Constants

- FAR_CLIP : double
- NEAR_CLIP : double

Fields

- anglesInDegrees : boolean
- bgColor : Color
- clipMatrix : Matrix
- depthBuffer : double[][]
- frameBuffer : Color[][]
- frameHeight : int
- frameWidth : int
- materials : String[]
- numOfMaterials : int
- objectBuffer : int[][]
- objectList : Object3D[]
- objectListLength : int
- refractiveIndices : double[]
- selectedObjID : int
- worldMaterial : int
- zoomX : double
- zoomY : double

Properties

«readOnly»
+ height : int
«readOnly»
+ orthographic : boolean
+ targetMaterial : int
«readOnly»
+ targetShape : String
«readOnly»
+ width : int

Constructors

+ Viewport(int, int) : void

Methods

+ addMaterial(String, double) : void
+ addRayBox(RayBox) : void
+ areAnglesInDegrees( ) : boolean
- calcClipMatrix( ) : void
- calcFaceColor(Vector, Vector, Vector, Object3D) : Color
- clearBuffers( ) : void
+ click(int, int) : void
+ getMaterials( ) : String[]
+ getNumOfMaterials( ) : int
+ getObjectList( ) : Object3D[]
+ getRefractiveIndices( ) : double[]
+ getWorldMaterial( ) : int
+ globallyRotateRayBox(double, double) : void
- inView(Object3D) : boolean
- initialiseMaterials( ) : void
- initialiseScene( ) : void
+ locallyRotateRayBox(double, double) : void
+ orbit(int, int) : void
- outlineSelectedObj( ) : void
+ paintComponent(Graphics) : void
- project(Vector) : Vector
- rasterise(Vector, Vector, Vector, Vector, double, Color, int) : void
- rasteriseFaceRow(int, int, int, double, Vector, double, Color, int) : void
- rasteriseHalfFace(int, int, double, double, double, double, double, Vector, double, Color, int) : void
+ recalculateBeams( ) : void
+ removeRayBox( ) : void
- render( ) : void
+ saveImage(File, String) : void
- setPixel(int, int, double, Color, int) : void
+ setTargetShape(Primitive) : void
+ setViewBack( ) : void
+ setViewBottom( ) : void
+ setViewFront( ) : void
+ setViewLeft( ) : void
+ setViewRight( ) : void
+ setViewTop( ) : void
+ setWorldMaterial(int) : void
+ toggleAngleUnits( ) : void
+ toggleOrthographic( ) : void
+ toggleShowAngles( ) : void
+ updateBeamThickness(int) : void
+ updateLabel(String) : void
- writeAngles(Graphics, FontRenderContext) : void
- writeRayBoxLabels(Graphics, FontRenderContext) : void
+ zoom(double) : void

Object

«memberClass»
RefractionSim::UIController::
TextFieldFocusListener

Fields

- actionListener : ActionListener
«final» «synthetic»
~ this$0 : UIController

Constructors

+ TextFieldFocusListener(UIController, ActionListener) : void

Methods

+ focusGained(FocusEvent) : void
+ focusLost(FocusEvent) : void

JFileChooser

«memberClass»
RefractionSim::UIController::
FileSaver

Fields

- gifDescription : String
- jpgDescription : String
- pngDescription : String
«final» «synthetic»
~ this$0 : UIController

Properties

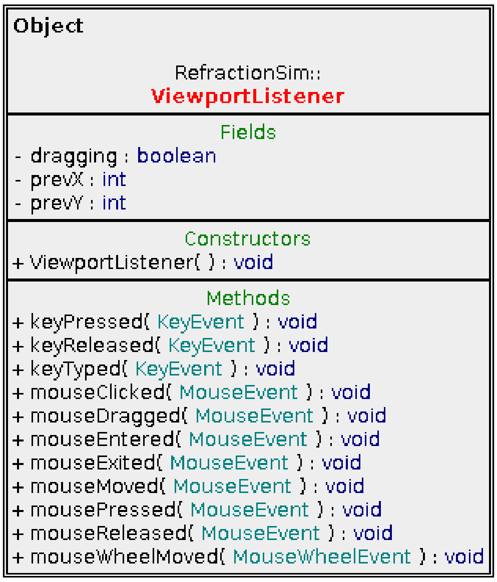
«readOnly»
+ fileWithExtension : File

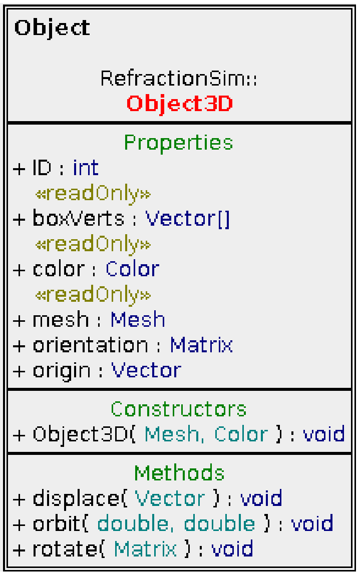
Constructors

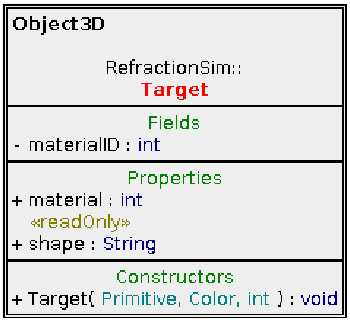
+ FileSaver(UIController) : void

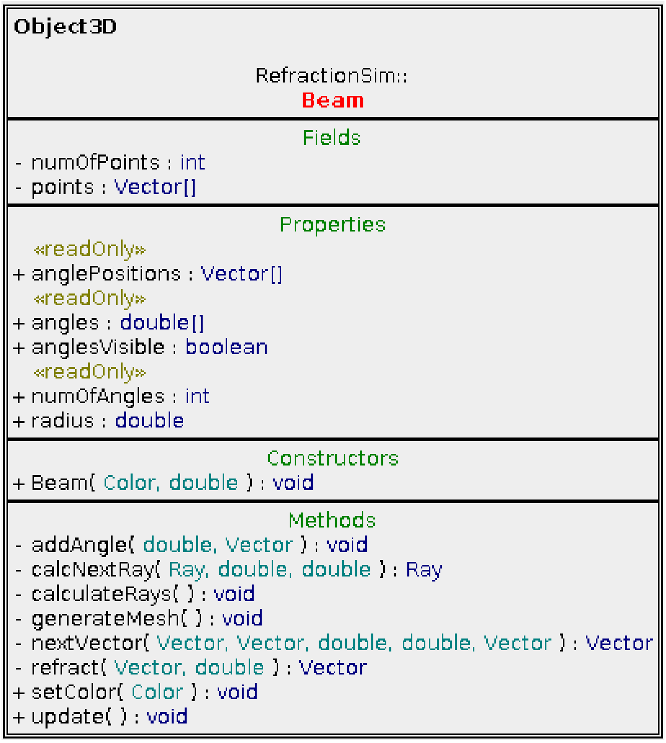
Methods

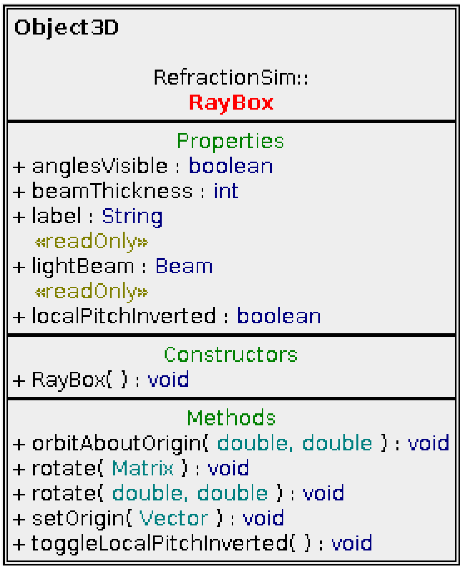
+ approveSelection( ) : void
+ extensionFromFileName(String) : String

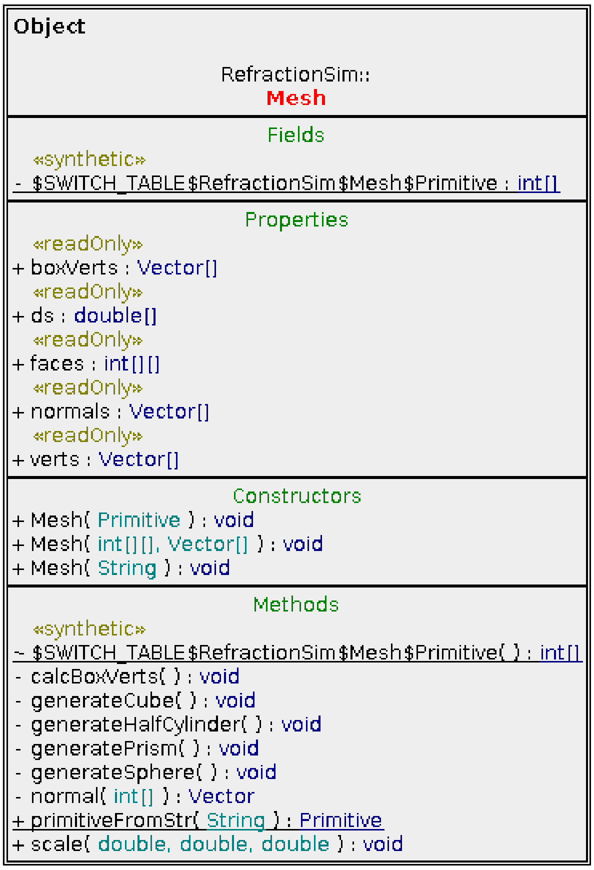

Object
RefractionSim::
ViewportListener
Fields
- dragging : boolean
- prevX : int
- prevY : int
Constructors
+ ViewportListener( ) : void
Methods
+ keyPressed(KeyEvent) : void
+ keyReleased(KeyEvent) : void
+ keyTyped(KeyEvent) : void
+ mouseClicked(MouseEvent) : void
+ mouseDragged(MouseEvent) : void
+ mouseEntered(MouseEvent) : void
+ mouseExited(MouseEvent) : void
+ mouseMoved(MouseEvent) : void
+ mousePressed(MouseEvent) : void
+ mouseReleased(MouseEvent) : void
+ mouseWheelMoved(MouseWheelEvent) : void

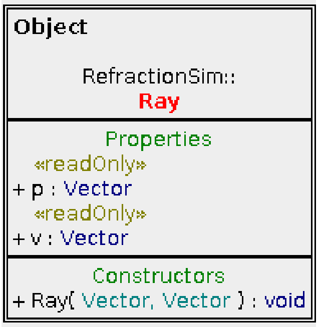

Object
RefractionSim::
Object3D
Properties
+ ID : int
«readOnly»
+ boxVerts : Vector[]
«readOnly»
+ color : Color
«readOnly»
+ mesh : Mesh
+ orientation : Matrix
+ origin : Vector
Constructors
+ Object3D(Mesh, Color) : void
Methods
+ displace(Vector) : void
+ orbit(double, double) : void
+ rotate(Matrix) : void

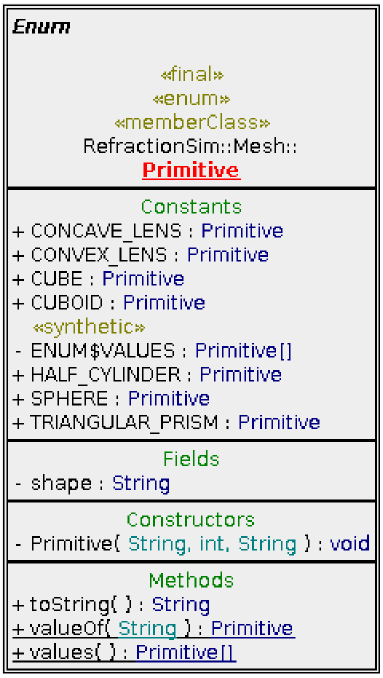

Object3D
RefractionSim::
Target
Fields
- materialID : int
Properties
+ material : int
«readOnly»
+ shape : String
Constructors
+ Target(Primitive, Color, int) : void

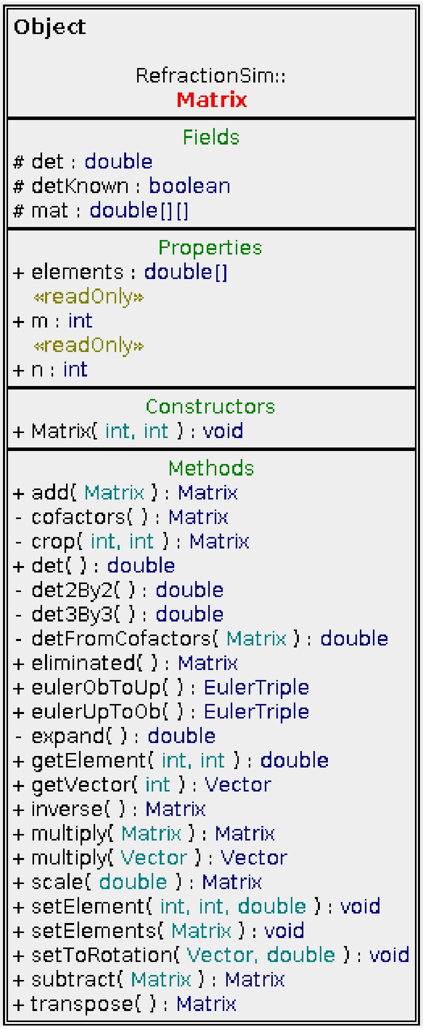

Object3D
RefractionSim::
Beam
Fields
- numOfPoints : int
- points : Vector[]
Properties
«readOnly»
+ anglePositions : Vector[]
«readOnly»
+ angles : double[]
+ anglesVisible : boolean
«readOnly»
+ numOfAngles : int
+ radius : double
Constructors
+ Beam(Color, double) : void
Methods
- addAngle(double, Vector) : void
- calcNextRay(Ray, double, double) : Ray
- calculateRays( ) : void
- generateMesh( ) : void
- nextVector(Vector, Vector, double, double, Vector) : Vector
- refract(Vector, double) : Vector
+ setColor(Color) : void
+ update( ) : void


Object3D
RefractionSim::
RayBox
Properties
+ anglesVisible : boolean
+ beamThickness : int
+ label : String
«readOnly»
+ lightBeam : Beam
«readOnly»
+ localPitchInverted : boolean
Constructors
+ RayBox( ) : void
Methods
+ orbitAboutOrigin(double, double) : void
+ rotate(Matrix) : void
+ rotate(double, double) : void
+ setOrigin(Vector) : void
+ toggleLocalPitchInverted( ) : void

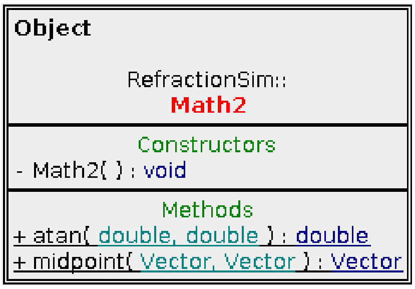

Object
RefractionSim::
Mesh
Fields
«synthetic»
- $SWITCH_TABLE$RefractionSim$Mesh$Primitive : int[]
Properties
«readOnly»
+ boxVerts : Vector[]
«readOnly»
+ ds : double[]
«readOnly»
+ faces : int[][]
«readOnly»
+ normals : Vector[]
«readOnly»
+ verts : Vector[]
Constructors
+ Mesh(Primitive) : void
+ Mesh(int[][], Vector[]) : void
+ Mesh(String) : void
Methods
«synthetic»
~ $SWITCH_TABLE$RefractionSim$Mesh$Primitive( ) : int[]
- calcBoxVerts( ) : void
- generateCube( ) : void
- generateHalfCylinder( ) : void
- generatePrism( ) : void
- generateSphere( ) : void
- normal(int[]) : Vector
+ primitiveFromStr(String) : Primitive
+ scale(double, double, double) : void


Object
RefractionSim::
Ray
Properties
«readOnly»
+ p : Vector
«readOnly»
+ v : Vector
Constructors
+ Ray(Vector, Vector) : void

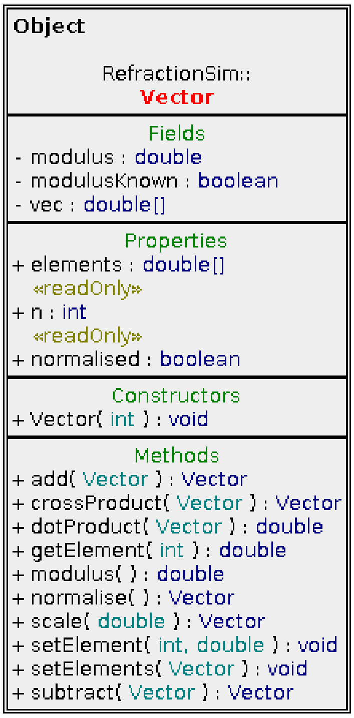

Enum
«final»
«enum»
«memberClass»
RefractionSim::Mesh::
Primitive
Constants
+ CONCAVE_LENS : Primitive
+ CONVEX_LENS : Primitive
+ CUBE : Primitive
+ CUBOID : Primitive
«synthetic»
- ENUM$VALUES : Primitive[]
+ HALF_CYLINDER : Primitive
+ SPHERE : Primitive
+ TRIANGULAR_PRISM : Primitive
Fields
- shape : String
Constructors
- Primitive(String, int, String) : void
Methods
+ toString( ) : String
+ valueOf(String) : Primitive
+ values( ) : Primitive[]



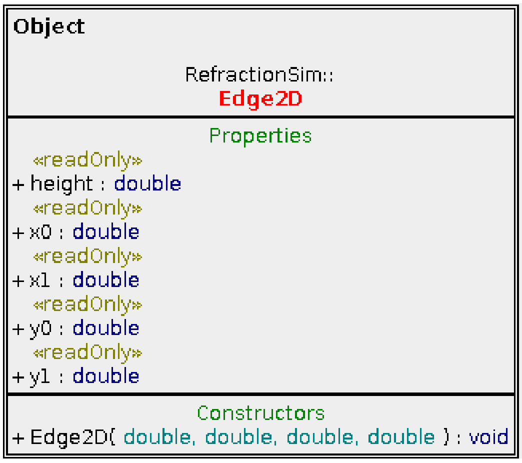
```
Object
RefractionSim::
Matrix
Fields
# det : double
# detKnown : boolean
# mat : double[][]
Properties
+ elements : double[]
  «readOnly»
+ m : int
  «readOnly»
+ n : int
Constructors
+ Matrix( int, int ) : void
Methods
+ add( Matrix ) : Matrix
- cofactors( ) : Matrix
- crop( int, int ) : Matrix
+ det( ) : double
- det2By2( ) : double
- det3By3( ) : double
- detFromCofactors( Matrix ) : double
+ eliminated( ) : Matrix
+ eulerObToUp( ) : EulerTriple
+ eulerUpToOb( ) : EulerTriple
- expand( ) : double
+ getElement( int, int ) : double
+ getVector( int ) : Vector
+ inverse( ) : Matrix
+ multiply( Matrix ) : Matrix
+ multiply( Vector ) : Vector
+ scale( double ) : Matrix
+ setElement( int, int, double ) : void
+ setElements( Matrix ) : void
+ setToRotation( Vector, double ) : void
+ subtract( Matrix ) : Matrix
+ transpose( ) : Matrix
```



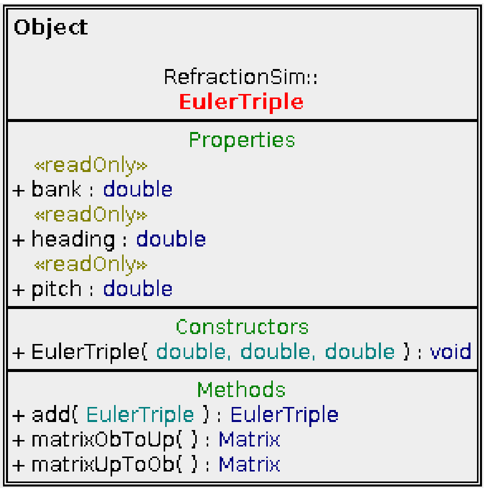
```
Object
RefractionSim::
Math2
Constructors
- Math2( ) : void
Methods
+ atan( double, double ) : double
+ midpoint( Vector, Vector ) : Vector
```



```
Object
RefractionSim::
Vector
Fields
- modulus : double
- modulusKnown : boolean
- vec : double[]
Properties
+ elements : double[]
  «readOnly»
+ n : int
  «readOnly»
+ normalised : boolean
Constructors
+ Vector( int ) : void
Methods
+ add( Vector ) : Vector
+ crossProduct( Vector ) : Vector
+ dotProduct( Vector ) : double
+ getElement( int ) : double
+ modulus( ) : double
+ normalise( ) : Vector
+ scale( double ) : Vector
+ setElement( int, double ) : void
+ setElements( Vector ) : void
+ subtract( Vector ) : Vector
```

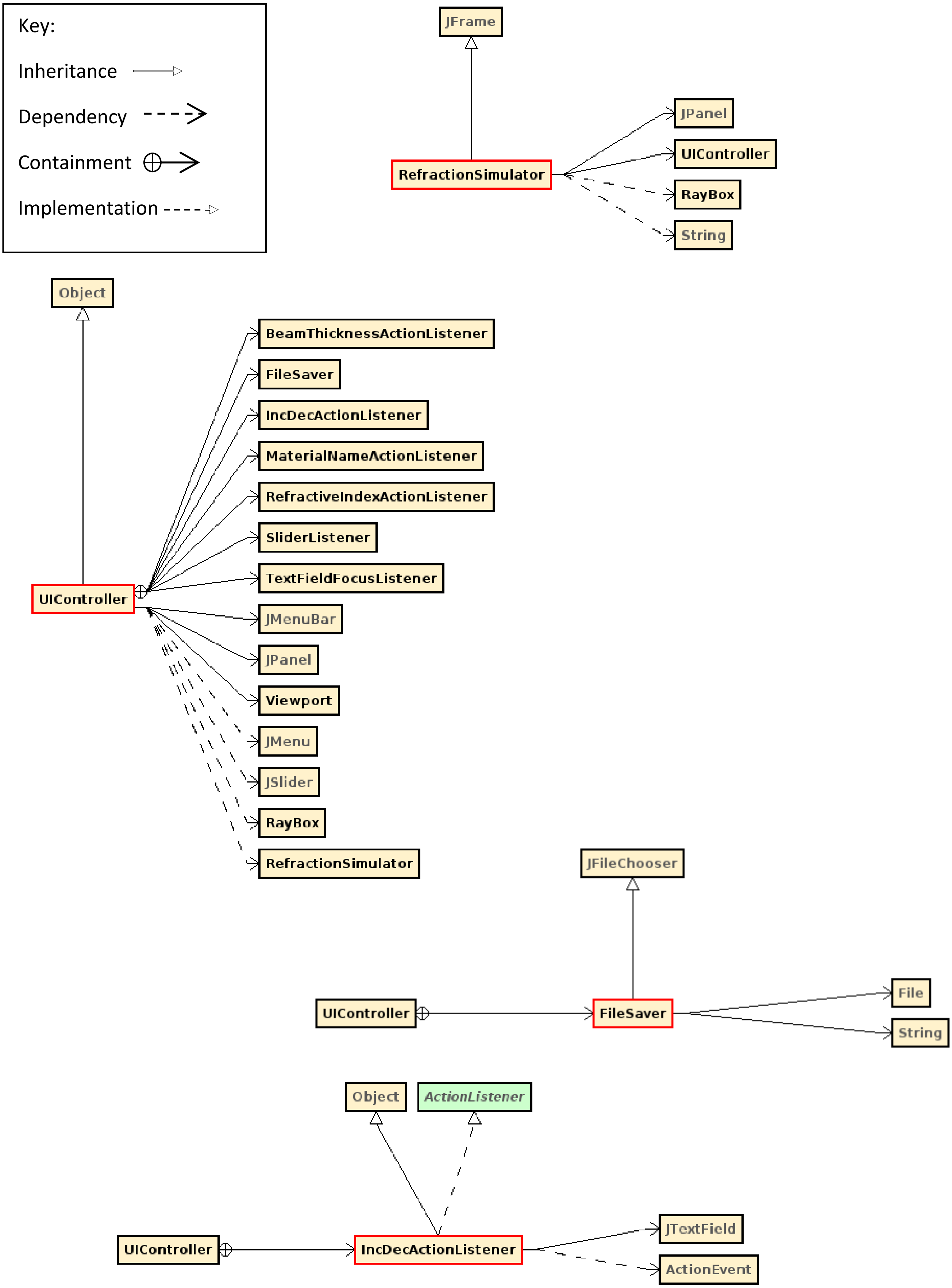


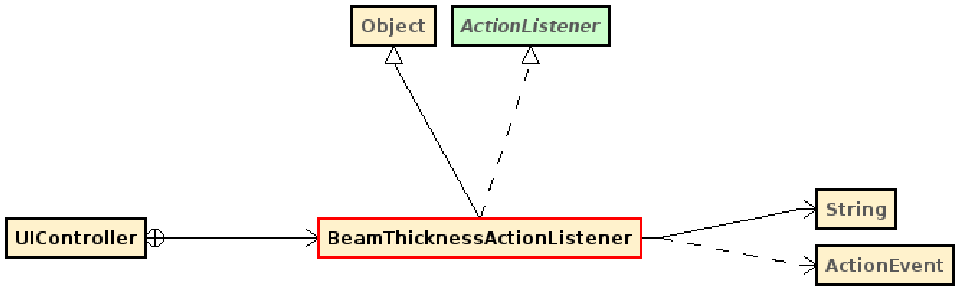
```
Object
RefractionSim::
Edge2D
Properties
  «readOnly»
+ height : double
  «readOnly»
+ x0 : double
  «readOnly»
+ x1 : double
  «readOnly»
+ y0 : double
  «readOnly»
+ y1 : double
Constructors
+ Edge2D( double, double, double, double ) : void
```

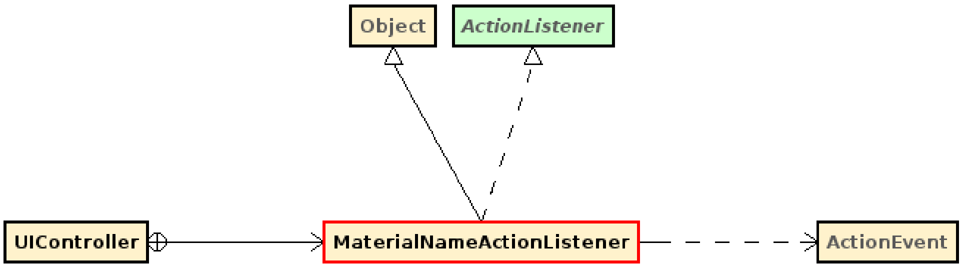


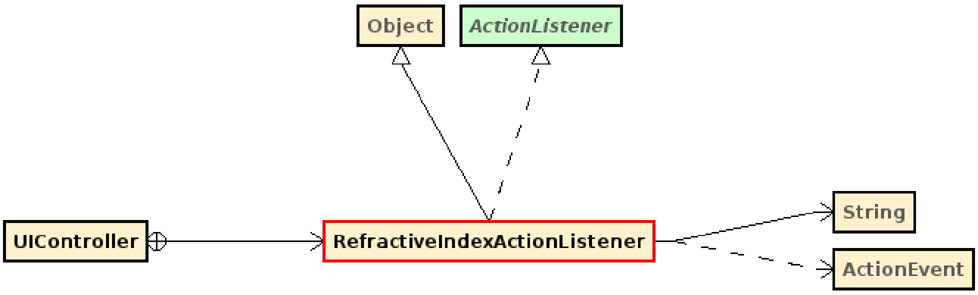
```
Object
RefractionSim::
EulerTriple
Properties
  «readOnly»
+ bank : double
  «readOnly»
+ heading : double
  «readOnly»
+ pitch : double
Constructors
+ EulerTriple( double, double, double ) : void
Methods
+ add( EulerTriple ) : EulerTriple
+ matrixObToUp( ) : Matrix
+ matrixUpToOb( ) : Matrix
```

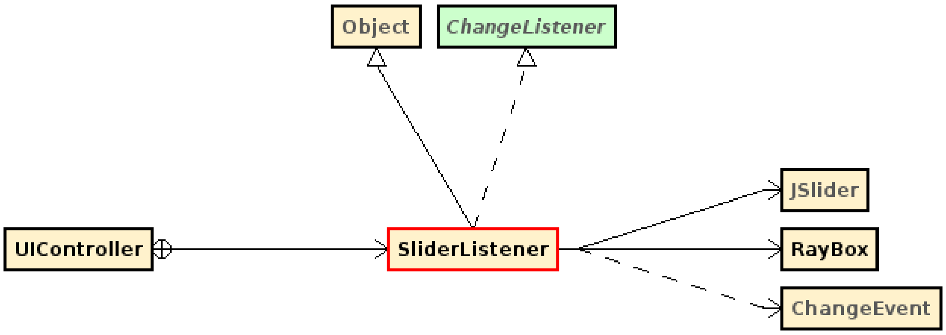
5.1.3 Class Relationships

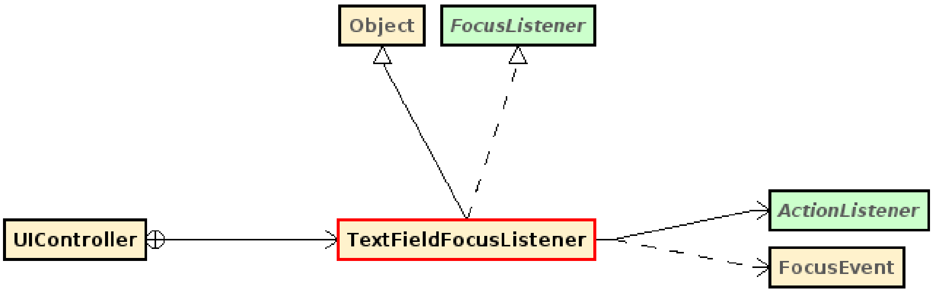


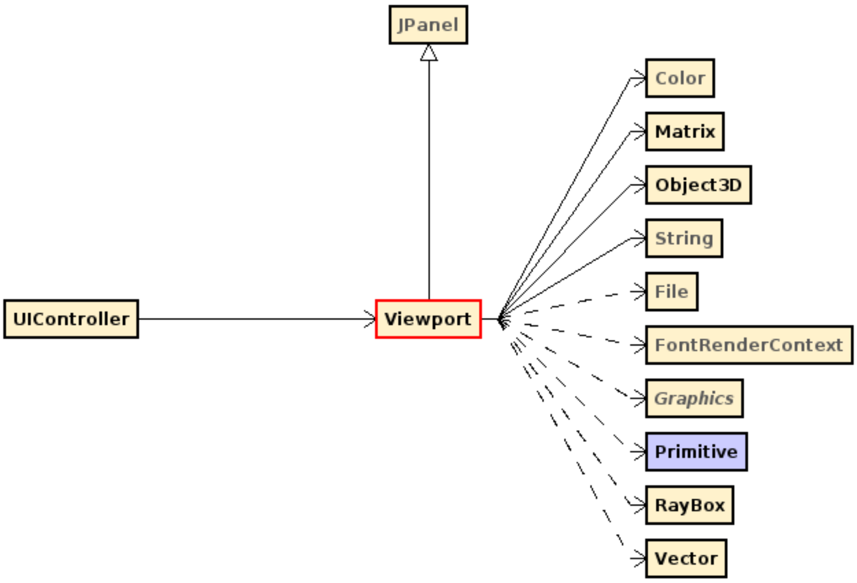

Object
ActionListener
UIController
BeamThicknessActionListener
String
ActionEvent

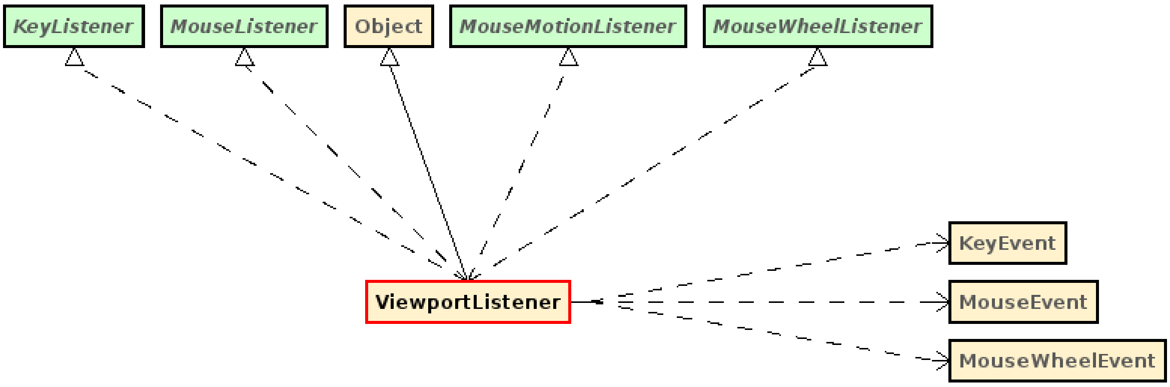

Object
ActionListener
UIController
MaterialNameActionListener
ActionEvent

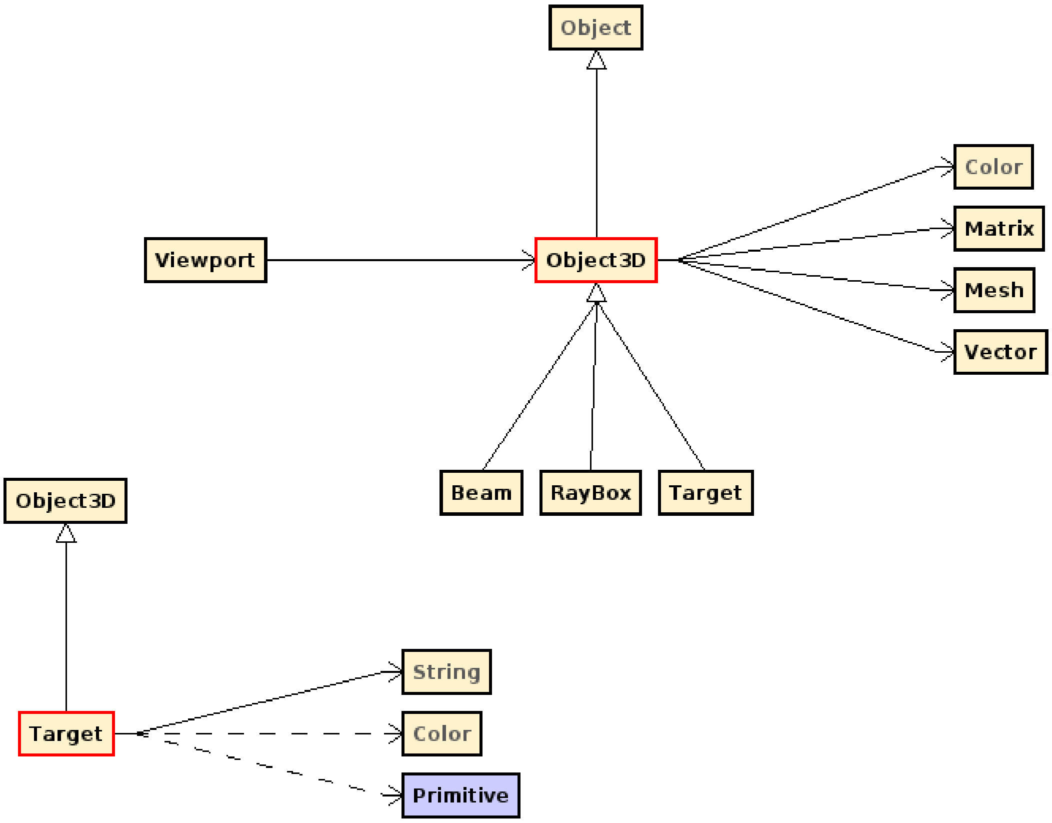

Object
ActionListener
UIController
RefractiveIndexActionListener
String
ActionEvent

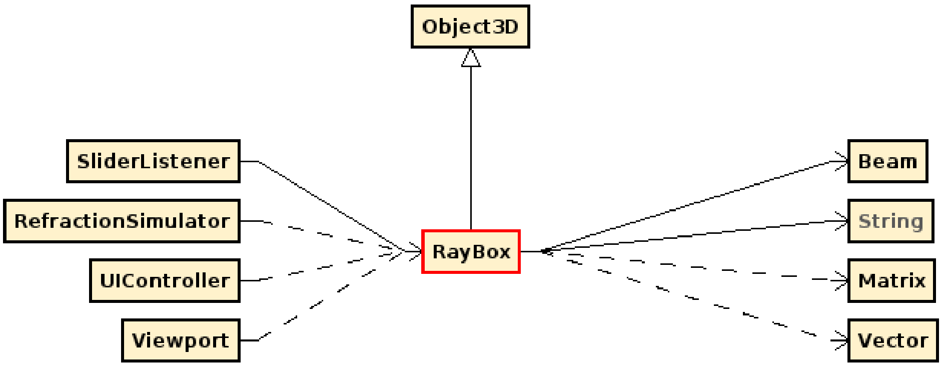

Object
ChangeListener
UIController
SliderListener
JSlider
RayBox
ChangeEvent

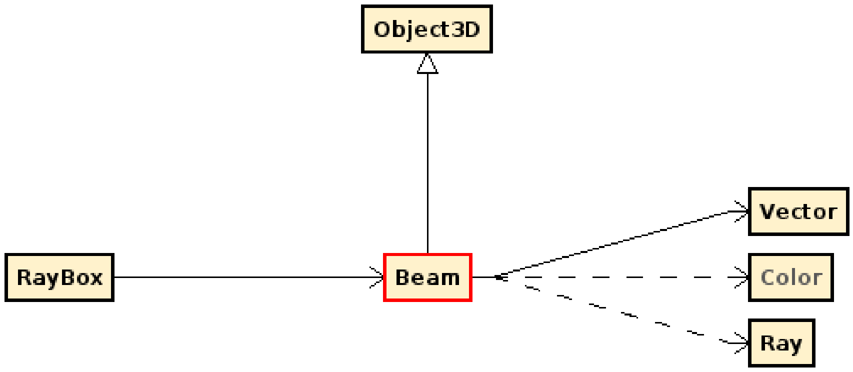

Object
FocusListener
UIController
TextFieldFocusListener
ActionListener
FocusEvent

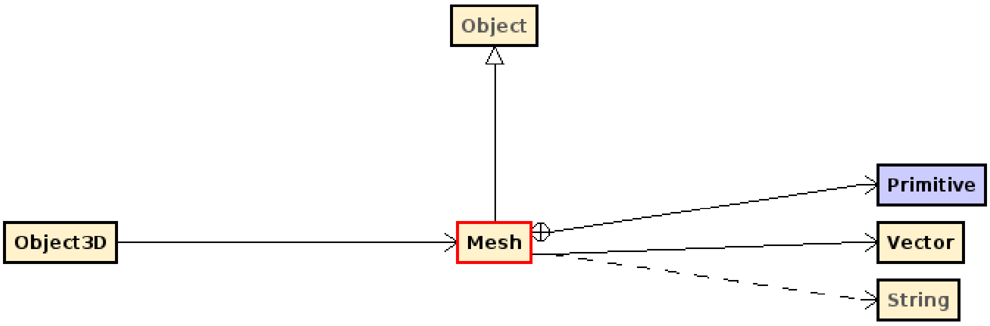

JPanel
Color
Matrix
Object3D
String
UIController
Viewport
File
FontRenderContext
Graphics
Primitive
RayBox
Vector

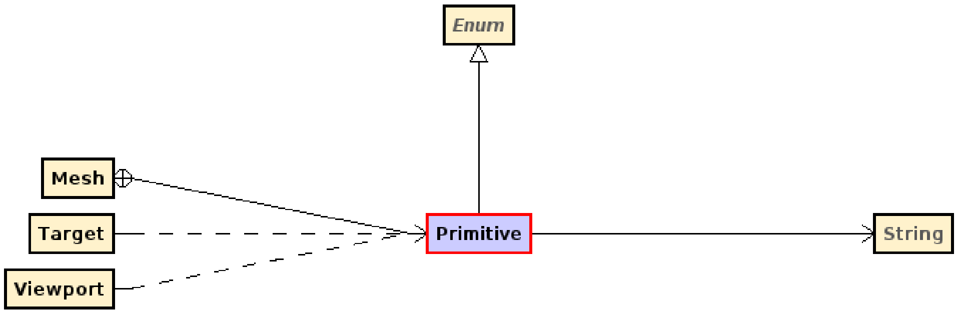

KeyListener
MouseListener
Object
MouseMotionListener
MouseWheelListener
KeyEvent
ViewportListener
MouseEvent
MouseWheelEvent

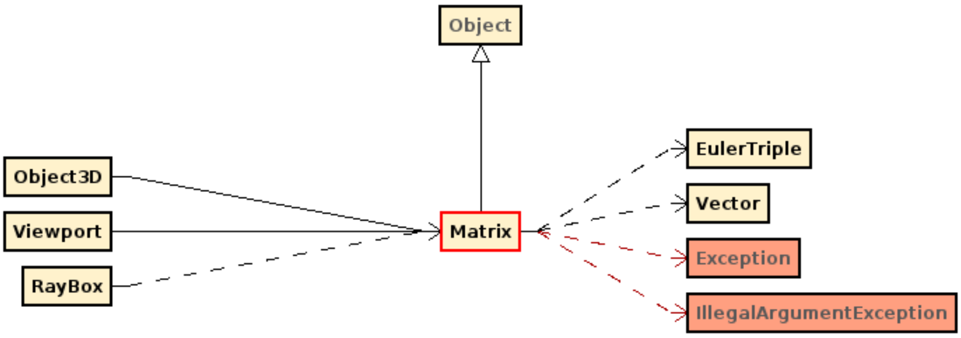

Object
Color
Viewport
Object3D
Matrix
Mesh
Vector
Beam
RayBox
Target
Object3D
String
Target
Color
Primitive

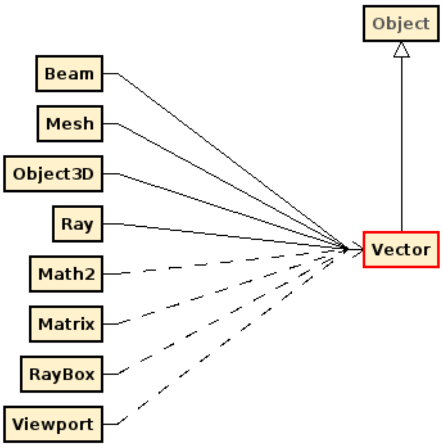

Object3D
SliderListener
RefractionSimulator
UIController
Viewport
RayBox
Beam
String
Matrix
Vector

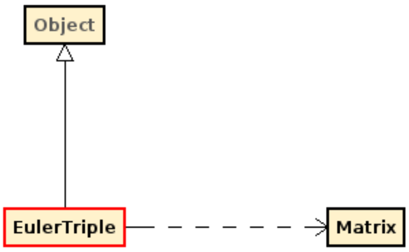

Object3D
RayBox
Beam
Vector
Color
Ray

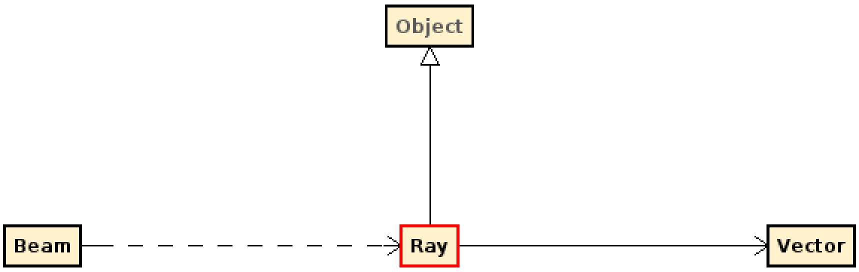

Object
Object3D
Mesh
Primitive
Vector
String

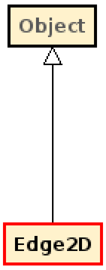

Enum
Mesh
Target
Viewport
Primitive
String

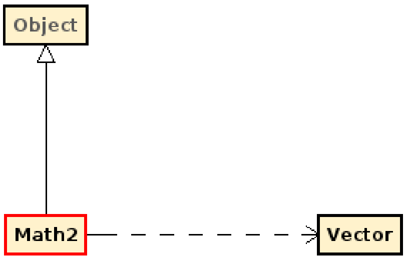

Object
Object3D
Viewport
RayBox
Matrix
EulerTriple
Vector
Exception
IllegalArgumentException


Object
Beam
Mesh
Object3D
Ray
Math2
Matrix
RayBox
Viewport
Vector

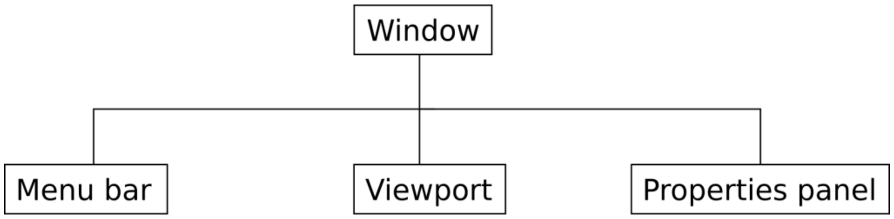

Object
EulerTriple
Matrix

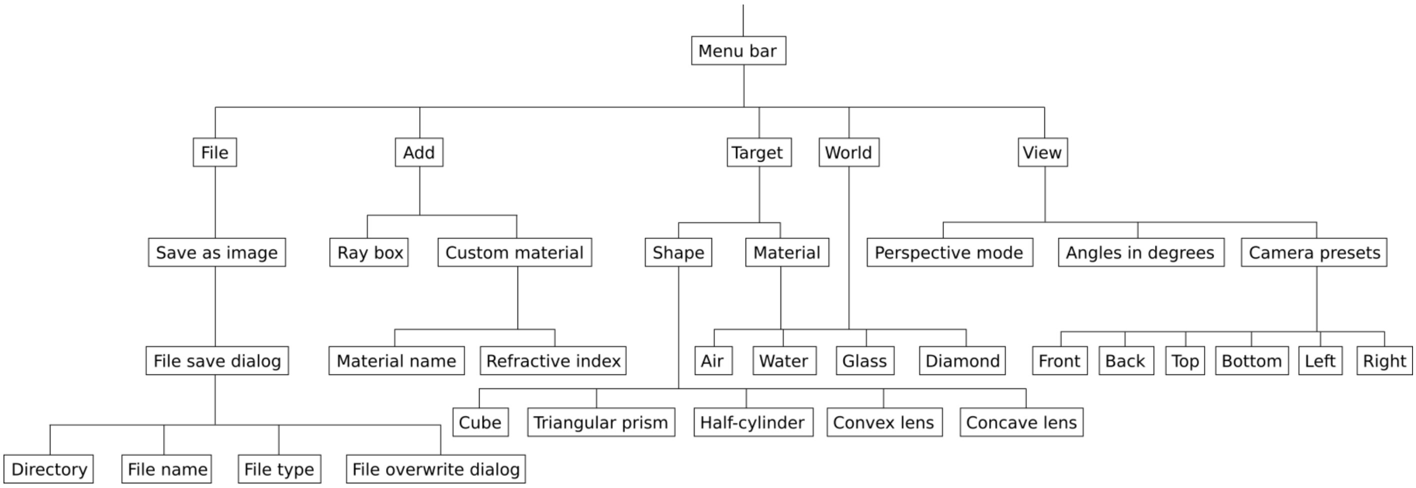

Object
Beam
Ray
Vector

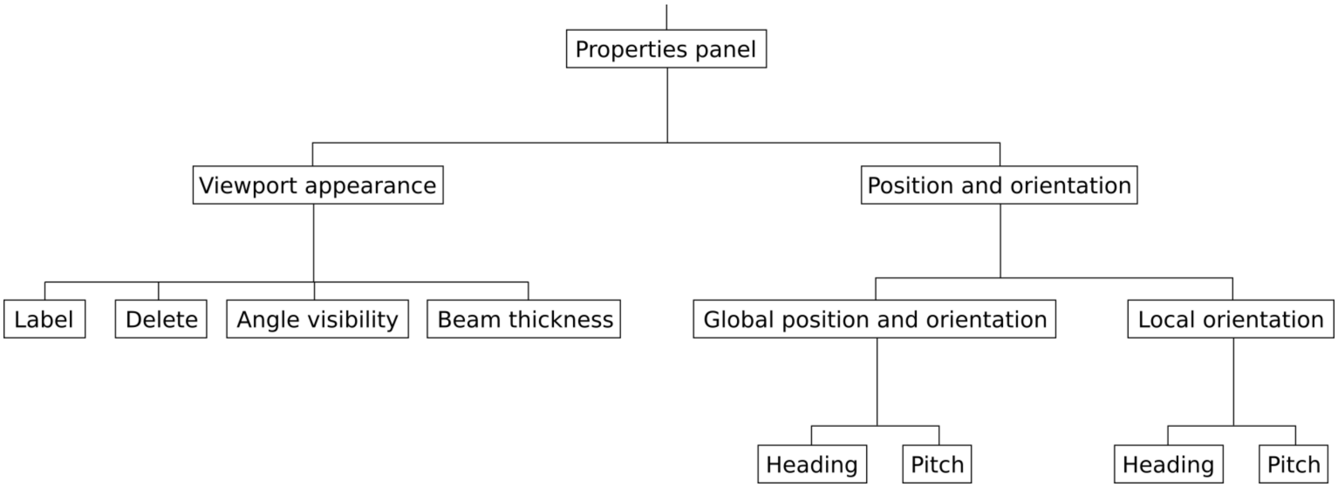

Object
Edge2D

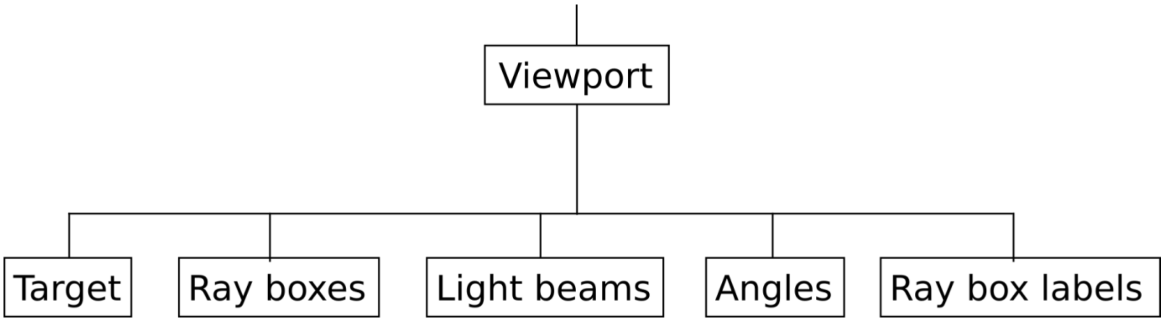

Object
Math2
Vector

5.1.4 User Interface Structure





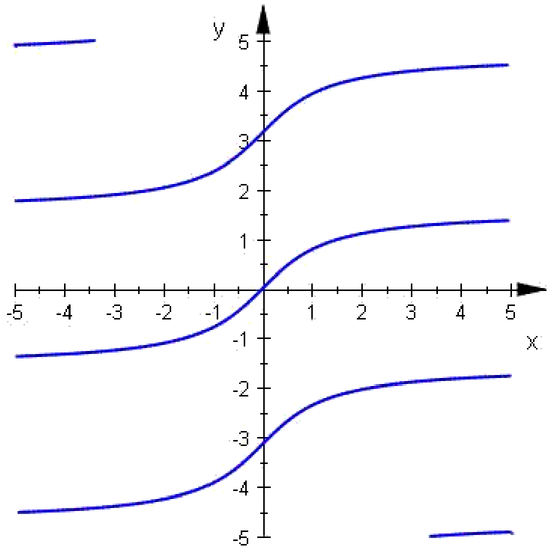




5.2 Algorithm Explanations

5.2.1 Extended Arctan

Java already provides an `atan` function as part of the `Math` class, but this restricts the range of arctan (x) to $\frac{\pi}{2} < x < \frac{\pi}{2}$ (returning only the principal root) so that it has one-to-one mapping and is a function in mathematical terms. In practice, we usually have angles in the interval $(-\pi, \pi]$ because this allows us to express any point circle. The tan (x) function has an infinite domain (the input can be any real number) and repeats with a period of π, so its inverse can be used to give angles in the range $(-\pi, \pi]$ if it is not limited to being a function in the mathematical sense. However, there are two possible angles and it is not possible to determine the one that is desired from this single input.

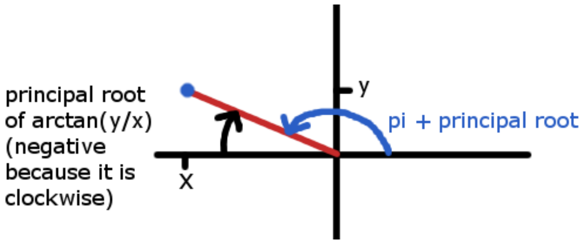


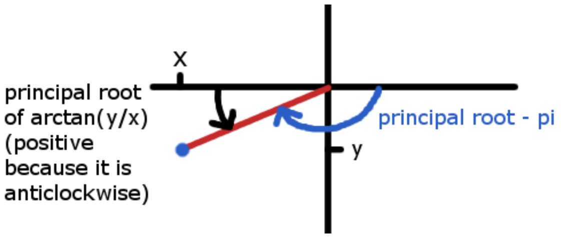
a graph of y = arctan(x) where y is in radians and arctan is not a mathematical function (image from matmin.kevius.com/trigpic/arctan1.gif)

The method I have developed as part of the `Math2` class takes two inputs: the y co-ordinate of a point and the x co-ordinate of that point. The value returned is the angle of the point anticlockwise from the positive x-axis. The method utilises Math.atan, but also offsets the result by $\pm\pi$ and deals with the special cases when the x co-ordinate is 0 (as this would cause a divided by zero and represents asymptotes on the graph).

5.2.1.1 Pseudocode

```
get point
if point is on the y-axis
    if point is at the origin
        return 0 radians
    else if point is on the positive y-axis
        return pi/2
    else (i.e. when the point is on the negative y-axis)
        return -pi/2
else if point is to the right of the y-axis
    return principal root of arctan(y co-ordinate / x co-ordinate)
else (i.e. when the point is to the left of the y-axis)
    if point is above the x-axis
        return pi + principal root of arctan(y / x)
    else (i.e. when the point is in the bottom left quadrant)
        return -pi + principal root of arctan(y / x)
```


principal root of arctan(y/x) (negative because it is clockwise)
y
x
pi + principal root


principal root of arctan(y/x) (positive because it is anticlockwise)
x
y
principal root - pi

5.2.1.2 Java Code

```java
public static double atan(double y, double x) {
	if (x == 0) {
		if (y == 0) {
			return 0;
		} else if (y > 0) {
			return (Math.PI / 2);
		} else {
			return (-Math.PI / 2);
		}
	} else if (x > 0) {
		return Math.atan(y / x);
	} else {
		if (y >= 0) {
			return (Math.atan(y / x) + Math.PI);
		} else {
			return (Math.atan(y / x) - Math.PI);
		}
	}
}
```

5.2.2 Check Whether AOBB is in View

To save time when rendering, objects can be ignored if they are not going to be in the user's/camera's view. However, checking if an object is in view could still be quite time consuming if there are many vertices, hence it is checked if the object's arbitrarily orientated bounding box (AOBB) is in view of the camera, as this can only ever have six vertices. If the bounding box is not in view, then the object can't be in view either, otherwise the object is likely to be in view.

To perform this task it is not simply enough to check if any of the vertices are in view, as there could be a face in view comprised of three out of view vertices, leading to a false negative. Instead, the method I used treats the x, y and z axes of normalised clip space independently, looking for a point in the viewable range or two points on opposite sides of the range. If, for each of the cardinal axes, there is a point in the viewable range or two points spanning the viewable range, the function returns true. This could give some false positives, but these are quite rare and are preferable to false negatives which would mean an object isn't rendered when it should be.

5.2.2.1 Pseudocode

```
get upper and lower bounds for each axis
inView ← false
for each axis (x, y and z)
      set below[axis], above[axis] and span[axis] to false
for each vertex of the AOBB while inView = false
      map vertex to camera space
      project vertex to normalised clip space
      for each axis
            if span[axis] = false
                  if z component of vertex is above lower bound for axis
                        if z component is below upper bound for axis
                              span[axis] ← true
                        else
                              above[axis] ← true
                  else
                        below[axis] ← true
                  if below[axis] = true and above[axis] = true
                        span[axis] ← true
                  if span[x] and span[y] and span[z]
                        inView ← true
```

5.2.2.2 Java Code

```
private boolean inView(Object3D obj) { // Even if this returns true, the
object may not be in view, as this is a quick algorithm
            boolean inView = false;
            // For each array variable, index 0 represents the x-axis,
index 1 represents the y-axis and index 2 represents the z-axis
            boolean[] below = {false, false, false};
            boolean[] above = {false, false, false};
            boolean[] span = {false, false, false}; // Each span element is
set to true if there is a normalised clip space point in the visible range
or one point above the range and one point below the range
            int[][] bounds = {{-1, 1}, {-1, 1}, {0, 1}}; // For each inner
array, index 0 is the lower bound for that axis and index 1 is the upper
bound
            Matrix objectToUpright = obj.getOrientation(); // Matrix for
transforming points from object space to upright space
            Matrix uprightToCamera =
objectList[0].getOrientation().transpose(); // Matrix for transforming
points from the camera's upright space to the camera's object space (camera
space)
            Vector[] boxVerts = obj.getBoxVerts();
            int i = 0;
            while ((i < boxVerts.length) && (!inView)) {
                  Vector worldCoord =
objectToUpright.multiply(boxVerts[i]).add(obj.getOrigin()); // Map the
point in object space to world space (via upright space)
                  Vector cameraCoord =
uprightToCamera.multiply(worldCoord.subtract(objectList[0].getOrigin()));
// Map the point in world space to camera space (via the camera's upright
space)
                  Vector normalisedCoord = project(cameraCoord);
                  for (int j = 0; (j < 3) && (inView == false); j++) { //
For each of the x, y and z axes
                        if (span[j] == false) {
                              if (normalisedCoord.getElement(j) >=
bounds[j][0]) {
                                    if (normalisedCoord.getElement(j) <=
bounds[j][1]) {
                                          span[j] = true; // A point within
the bounds is treated like one on each side because both contribute to
making the box visible on that axis
                                    } else {
                                          above[j] = true;
                                    }
                              } else {
                                    below[j] = true;
                              }
                              if ((below[j]) && (above[j])) {
                                    span[j] = true;
                              }
                              if ((span[0]) && (span[1]) && (span[2])) {
                                    inView = true;
                              }
                        }
                  }
                  i++;
            }
            return inView;
      }
```

5.2.3 Validating Beam Thickness Input

This method (in the `BeamThicknessActionListener` class) is called when the beam thickness text field loses focus, gains focus, or enter is hit when it is in focus. Its purpose is to ensure that the value in the field is an integer between 1 and 10 inclusive. This is achieved by reverting non-numerical input to the last valid value (including adjusted invalid numerical inputs) that was entered – this is the default value if this is the first value entered. Non-integers are rounded to the nearest integer and if the result or original integer is not in the valid range it is rounded to the closest integer in the valid range; this means that values below 1 become 1 and those above 10 become 10. If the input was numerical, then the adjusted version is stored for the next time this method is called so that it can be used as the last valid value. Furthermore, the beam's geometry must be regenerated to reflect the change in thickness.

5.2.3.1 Pseudocode

```
get input
if input is a number
      newBeamThickness ← input rounded to the nearest integer
      if newBeamThickness < 1
            newBeamThickness ← 1
      else if newBeamThickness > 10
            newBeamThickness ← 10
      set text field value to newBeamThickness
      update beam's geometry
      store newBeamThickness as the last valid value
else
      set text field's value to the last valid value
```

5.2.3.2 Java Code

```
public void actionPerformed(ActionEvent event) {
      JTextField source = (JTextField)(event.getSource());
      try {
            int newBeamThickness =
(int)(Math.round(Double.parseDouble(source.getText()))); // Round to the
nearest integer
            // Values below 1 become 1 and those above 10 become 10
            if (newBeamThickness < 1) {
                  newBeamThickness = 1;
            } else if (newBeamThickness > 10) {
                  newBeamThickness = 10;
            }
            String newText = Integer.toString(newBeamThickness);
            source.setText(newText);
            viewport.updateBeamThickness(newBeamThickness);
            lastValid = newText; // Store the most recent valid entry so
that an non-numerical entry can be reverted to this
      } catch (Exception e) { // If the contents of the text field is non-
numerical (causing parseDouble() to throw an exception)
            source.setText(lastValid); // Revert to the last valid entry
      }
}
```

5.3 Descriptions of Methods

public class **RefractionSim.Beam** extends RefractionSim.Object3D

Class for light beams

Constructors

public Beam(
Color color,
double radius)

Constructor for the Beam class that sets its colour, radius, default position and orientation, and sets angles to be displayed in the viewport

Parameters

color - the colour of the light beam (the beam will be a solid colour and not have shading to imitate lighting)

radius - half of the width of the square cross-section of the beam

Methods

public void setColor(
Color newColor)

Sets the colour of the beam to newColor

Parameters

newColor - the new colour for the beam

public void update()

Recalculates the beam's path and regenerates its geometry

public double getRadius()

Returns half of the width of the square cross-section of the beam

Returns

half of the width of the square cross-section of the beam

public void setRadius(
double newRadius)

Sets the width of the square cross-section of the beam to be 2 * newRadius and regenerates the geometry of the beam

Parameters

newRadius - half of the width of the new square cross-section of the beam

public boolean getAnglesVisible()

Returns whether or not angles are set to be visible for the light beam

Returns

if angles are to be visible

public void setAnglesVisible(
boolean showAngles)

Sets whether or not angles should be displayed for the light beam

Parameters

showAngles - if angles are to be visible

public double[] getAngles()

Returns the list of angles in order from the ray box

Returns

the list of angles between the beam and the surface normals in order from the ray box

public RefractionSim.Vector[] getAnglePositions()

Returns the list of positions in 3-D space for each of the angles in order from the ray box

Returns

the list of positions for the angles in the same order as the list of angles

public int getNumOfAngles()

Returns the number of angles in the list of angles

Returns

the number of angles in the list of angles

private void generateMesh()

Generates the geometry of the beam if the rays have been calculated

private void calculateRays()

Calculates the path of the beam as a sequence of rays and stores the points in 3-D space where the path switches between rays and the angles of rays to surface normals

private RefractionSim.Ray calcNextRay(
Ray incidentRay,
double targetIndexRelToWorld,
double criticalAngle)

Calculates and returns the next ray of the beam based on the intersection of the current ray and the target object

Parameters

incidentRay - the last ray that was calculated

targetIndexRelToWorld - the refractive index of the target material relative to the world

criticalAngle - the minimum angle from the normal needed for total internal reflection within the denser material

Returns

the next ray which the beam follows

private RefractionSim.Vector nextVector(
Vector vector,
Vector normal,
double targetIndexRelToWorld,
double criticalAngle,
Vector intersection)

Calculates the direction of the next ray and returns the result as a normalised 3-row vector

Parameters

vector - the direction of the current ray

normal - the vector perpendicular to the face being intersected

targetIndexRelToWorld - the refractive index of the target material relative to the world

criticalAngle - the minimum angle between vector and normal within the denser material that would cause total internal reflection

intersection - the point of intersection with the face in world space

Returns

the direction (as a unit vector) of the next ray after the one with direction vector that intersects the face with normal normal at the point intersection

private RefractionSim.Vector refract(
Vector incidentVector,
double refractiveIndex)

Calculates and returns the vector produced by the refraction of incidentVector passing from material A to material B where refractiveIndex is the refractive index of material B relative to material A

Parameters

incidentVector - the direction of the previous ray rotated into the plane y = 0 with x-axis -normal

refractiveIndex - the refractive index of the destination material relative to the source material

Returns

the direction of the next ray

private void addAngle(
double angle,
Vector position)

Appends an angle to the list of angles and an angle position to the list of angle positions

Parameters

angle - the angle to append to the list

position - the position in 3-D space of the angle to append to the list

Fields

private points

private numOfPoints

private angles

private anglePositions

private numOfAngles

private radius

private anglesVisible

public class **RefractionSim.Edge2D**

Class for edges (line segments) in two dimensions

Constructors

public Edge2D(
double x0,
double y0,
double x1,
double y1)

Constructor for the Edge2D class

Parameters

x0 - the x co-ordinate of the first end of the edge
y0 - the y co-ordinate of the first end of the edge
x1 - the x co-ordinate of the second end of the edge
y1 - the y co-ordinate of the second end of the edge

Methods

public double getX0()

Returns the x co-ordinate of the lower end of the edge

Returns

the x co-ordinate of the lower of the two end points of the edge

public double getX1()

Returns the x co-ordinate of the higher end of the edge

Returns

the x co-ordinate of the higher of the two end points of the edge

public double getY0()

Returns the y co-ordinate of the lower end of the edge

Returns

the y co-ordinate of the lower of the two end points of the edge

public double getY1()

Returns the y co-ordinate of the higher end of the edge

Returns

the y co-ordinate of the higher of the two end points of the edge

public double getHeight()

Returns the vertical height of the edge

Returns

the y component of the edge's length

Fields

private x0

private y0

private x1

private y1

private height

public class **RefractionSim.EulerTriple**

Class for Euler angle triples (these represent an orientation in 3-D space)

Constructors

public EulerTriple(
double heading,
double pitch,
double bank)

Constructor for the EulerTriple class which sets the three angles to the three parameters

Parameters

heading - the rotation about the vertical (y) axis
pitch - the rotation about the object space x-axis after applying the heading rotation
bank - the rotation about the object space z-axis after applying the heading and pitch rotations

Methods

public double getHeading()

Returns the heading angle

Returns

the angle of rotation about the vertical axis

public double getPitch()

Returns the pitch angle

Returns

the angle of declination

public double getBank()

Returns the bank angle

Returns

the angle of rotation along the body z-axis

public RefractionSim.Matrix matrixObToUp()

Returns the matrix for transforming points from object space to upright space where the EulerTriple is the angular displacement of object space from upright space

Returns

the object space to upright space matrix represented by the EulerTriple

public RefractionSim.Matrix matrixUpToOb()

Returns the matrix for transforming points from upright space to object space where the EulerTriple is the angular displacement of object space from upright space

Returns

the upright space to object space matrix represented by the EulerTriple

public RefractionSim.EulerTriple add(
EulerTriple toAdd)

Returns the sum of the EulerTriple for which this method is called and the toAdd parameter

Parameters

toAdd - representation of an angular displacement to add to the angular displacement represented by the EulerTriple for which this method is called

Returns

the EulerTriple representing the sum of the two angular displacements

Fields

private heading

private pitch

private bank

public class **RefractionSim.Math2**

Class for mathematical functions which are not already provided in Math.

Constructors

private Math2()

The constructor for the Math2 class is private because there aren't supposed to be any instances of this class. The purpose of this class is the provide publicly accessible static methods like the Math class

Methods

public static double atan(
double y,
double x)

Returns an angle (in radians) from a vertical component and a horizontal component. All vectors in the plane can be given a proper angle by this function; Math.atan limits angles to the interval (-pi/2, pi/2)

Parameters

y - the vertical component of a 2-D vector
x - the horizontal component of a 2-D vector

Returns

the angle (in radians) of a vector to the positive x-axis under standard mathematical conventions

public static RefractionSim.Vector midpoint(
Vector p0,
Vector p1)

Returns the midpoint of the line segment between p0 and p1 in n-dimensional space

Parameters

p0 - one end of a line segment
p1 - the other end of the line segment

Returns

the point on the line between p0 and p1 that is halfway between the two points

public class **RefractionSim.Matrix**

Class for two-dimensional column-major (each column is a vector) matrices

Constructors

public Matrix(
int m,
int n)

Constructor for the Matrix class which sets all elements in the new matrix to zero

Parameters

m - number of columns that the new matrix is to have
n - number of rows that the new matrix is to have

Throws

IllegalArgumentException - if either m or n is less than 2 (as this would produce a vector or single value)

Methods

public int getM()

Returns the number of columns the matrix has

Returns

the number of columns the matrix has

public int getN()

Returns the number of rows the matrix has

Returns

the number of rows the matrix has

public void setElement(
int i,
int j,
double newValue)

Sets the value of a single element in the matrix to that of the newValue parameter

Parameters

i - the column of the element to change (indices start at 0)
j - the row of the element to change (indices start at 0)
newValue - the value which the specified element should be changed to

Throws

ArrayIndexOutOfBoundsException - if i >= number of columns or j >= number of rows

public void setElements(
double[] newValues)

Sets all elements of the matrix to the values in the newValues array (the array represents the matrix with columns joined end-to-end)

Parameters

newValues - the array of new values that the matrix elements should be set to; the matrix elements are changed down the columns starting with the leftmost column

Throws

IllegalArgumentException - if the length of the array is not equal to the number of elements in the matrix

public void setElements(
Matrix newValues)

Copies the values of the elements of the matrix newValues to the corresponding elements in the matrix for which this method is being called (the two matrices then do not reference the same memory locations)

Parameters

newValues - the matrix of new values that the matrix elements should be set to

Throws

IllegalArgumentException - if the newValues matrix does not have the same dimensions as the matrix for which this method is being called

public void setToRotation(
Vector axis,
double angle)

Sets the elements of the 3 by 3 matrix to represent an a specified 3-D angular displacement (rotation by a specific amount about a vector in 3-D space)

Parameters

axis - the 3-D vector/axis about which to rotate
angle - the number of radians by which to rotate

Throws

IllegalArgumentException - if the matrix is not 3 by 3 or if the axis is not a 3-row vector

public double getElement(
int i,
int j)

Gets the value of a single element in the matrix

Parameters

i - the column of the element to return (indices start at 0)
j - the row of the element to return (indices start at 0)

Returns

the value of the specified element

Throws

ArrayIndexOutOfBoundsException - if i >= number of columns or j >= number of rows or i or j < 0

public double[] getElements()

Returns the values of all elements in a one-dimensional array where the matrix's columns have been joined end-to-end

Returns

the array of values in the matrix where the columns have been joined end-to-end

public RefractionSim.Vector getVector(
int i)

Returns a column of the matrix as a Vector object (this program uses column-vectors)

Parameters

i - the index of the column to return as a Vector object (indices start at zero)

Returns

a Vector object representing the specified column of the matrix

Throws

IllegalArgumentException - if a the specified column does not exist

public RefractionSim.Matrix add(
Matrix toAdd)

Returns the sum of this matrix and the toAdd parameter matrix

Parameters

toAdd - the two-dimensional matrix to add to the matrix for which this method is being called

Returns

the sum of this matrix and the matrix passed as a parameter (the returned matrix will be the same size as both matrices being added)

Throws

IllegalArgumentException - if the toAdd matrix is not the same size as the matrix for which this method is being called

public RefractionSim.Matrix subtract(
Matrix toSubtract)

Returns A - B where A is the matrix for which this method is called and B is the toSubtract parameter matrix

Parameters

toSubtract - the matrix to subtract from the matrix for which this method is called

Returns

A - B where A is the matrix for which this method is called and B is the toSubtract parameter matrix

Throws

IllegalArgumentException - if the two matrices don't have the same dimensions

public RefractionSim.Matrix multiply(
Matrix toMultiply)

Returns the matrix AB, where A is the matrix for which this method is being called and B is the toMultiply parameter matrix

Parameters

toMultiply - the matrix to post-multiply by (toMultiply is pre-multiplied by the matrix for which this method is being called)

Returns

the matrix AB, where A is the matrix for which this method is being called and B is the toMultiply parameter

Throws

IllegalArgumentException - if the number of rows in the toMultiply matrix does not equal the number of columns in the matrix for which this method is being called

public RefractionSim.Vector multiply(
Vector toMultiply)

Returns the vector AB where A is the matrix for which this method is being called and B is the toMultiply parameter vector

Parameters

toMultiply - the vector to post-multiply by (toMultiply is pre-multiplied by the matrix for which this method is being called)

Returns

the vector AB, where A is the matrix for which this method is being called and B is the toMultiply parameter

Throws

IllegalArgumentException - if the number of rows in the toMultiply vector does not equal the number of columns in the matrix for which this method is being called

public RefractionSim.Matrix scale(
double scaleFactor)

Returns the matrix multiplied by a scalar value

Parameters

scaleFactor - the scalar value by which the matrix is to be multiplied

Returns

the matrix produced when the original matrix is multiplied by scaleFactor

public RefractionSim.Matrix inverse()

Returns the inverse of the matrix (if A is the original matrix and B is the inverse, AB = BA = I where I is the identity matrix (all elements are zero except for the diagonal from top left to bottom right on which the elements have the value 1)

Returns

the inverse of the matrix

Throws

Exception - if the matrix is not square (the number of columns equals the number of rows if a matrix is square) or the matrix has no inverse

public RefractionSim.Matrix transpose()

Returns the transpose of the matrix in which the columns of the original become the rows of the transpose and the rows of the original become the columns of the transpose (equivalent to reflecting the elements in the diagonal through the top left and bottom right elements of the matrix

Returns

the transpose of the matrix

public double det()

Returns the determinant of the matrix

Returns

the determinant of the matrix

Throws

IllegalArgumentException - if the matrix is not square (the number of columns equals the number of rows if a matrix is square)

private RefractionSim.Matrix cofactors()

Returns the matrix of cofactors corresponding to the matrix for which this method is being called

Returns

the matrix of cofactors corresponding to the matrix for which this method is being called

private double detFromCofactors(
Matrix cofactors)

An alternative to the det() method which resuses the cofactors calculated for the inverse method so that they don't need to be recalculated

Parameters

cofactors - the matrix of cofactors corresponding to the matrix for which this method is being called

Returns

the determinant of the matrix for which this method is being called

Throws

IllegalArgumentException - if the matrix of cofactors doesn't have the same number of rows as the matrix for which this method is called

private double expand()

Recursive method for finding the determinant of a matrix by finding the determinants of sub-matrices

Returns

the determinant of the matrix for which the method is being called

public RefractionSim.Matrix eliminated()

Returns a version of the matrix where the columns have been manipulated and rows swapped such that the determinant is unchanged but calculation of the determinant through expanding of the top row is quicker (because the top row has many zeroes, so the determinant will be a sum of values of which many will be zero and we know which will be zero without having to do the full calculations)

Returns

a 'simpler' matrix with the same determinant as the original

private RefractionSim.Matrix crop(
int col,
int row)

Returns a sub-matrix (known as a minor) of the matrix for which this method is called by removing the column and the row specified

Parameters

col - the column to crop in order to create the sub-matrix
row - the row to crop in order to create the sub-matrix

Returns

the matrix for which the method is called but with a column and a row removed

private double det3By3()

Quick non-recursive method for calculating the determinant of a 3 by 3 matrix

Returns

the determinant of the matrix (must be 3 by 3)

private double det2By2()
Quick non-recursive method for calculating the determinant of a 2 by 2 matrix
Returns
the determinant of the matrix (must be 2 by 2)

public RefractionSim.EulerTriple eulerObToUp()
Returns the EulerTriple (three euler angles) representing the orientation of an object when this matrix is interpreted as converting points from an object space to the corresponding upright space
Returns
the orientation/angular displacement represented by the matrix when interpreted as converting points from object space to upright space in EulerTriple form
Throws
IllegalArgumentException - if the matrix is not 3 by 3 or if the matrix does not have determinant 1 (a matrix that satisfied these conditions is not necessarily a rotation matrix, so it is the responsibility of the code that calls this method to ensure the matrix represents a rotation)

public RefractionSim.EulerTriple eulerUpToOb()
Returns the EulerTriple (three euler angles) representing the orientation of an object when this matrix is interpreted as converting points from an upright space to the corresponding upright space
Returns
the orientation/angular displacement represented by the matrix when interpreted as converting points from object space to upright space in EulerTriple form
Throws
IllegalArgumentException - if the matrix is not 3 by 3 or if the matrix does not have determinant 1 (a matrix that satisfied these conditions is not necessarily a rotation matrix, so it is the responsibility of the code that calls this method to ensure the matrix represents a rotation)

Fields

private m

private n

private mat

private det

private detKnown

public class **RefractionSim.Mesh**

Class for the geometries of objects based on the object3D class and its descendant classes

Constructors

public Mesh(
int[][] faces,
Vector[] verts)
A constructor for the mesh class for non-primitive geometries
Parameters
faces - a list of the faces of the object; each item/face contains 3 items which are the indices of the vertices in the verts list that make up the face
verts - a list of vertices; each vertex is represented by a position in 3-D space

public Mesh(
Mesh.Primitive shape)
A constructor for the mesh class for primitive geometries
Parameters
shape - the shape represented by the new mesh
Throws
IllegalArgumentException - if shape is null

public Mesh(

String shape)

A constructor for the Mesh class for primitive geometries where the shape needs to be determined from its name

Parameters

shape - the name/String representation of the shape

Methods

public static RefractionSim.Mesh.Primitive primitiveFromStr(
String shape)

Returns a primitive shape as a Primitive object from its name/String representation

Parameters

shape - the name of the shape

Returns

the shape represented by the name

private void generateCube()

Creates the vertices and faces that define a cube of side length 2 units

private void generatePrism()

Creates the vertices and faces that define a triangular prism with sides of length 2

private void generateSphere()

Creates the vertices and faces that define an approximation of a sphere with radius 1

private void generateHalfCylinder()

Creates the vertices and faces that define the approximation of a cylinder of radius 1 and height 2 that has been cut in vertically in half

public void scale(
double xScale,
double yScale,
double zScale)

Stretches the geometry parallel to the object space axes

Parameters

xScale - the scale factor of enlargement parallel to the x-axis
yScale - the scale factor of enlargement parallel to the y-axis
zScale - the scale factor of enlargement parallel to the z-axis

public RefractionSim.Vector[] getBoxVerts()

Returns a list of the vertices for the mesh's arbitrarily orientated bounding box (AOBB)

Returns

the AOBB vertices for the geometry

private void calcBoxVerts()

Calculates the vertices for the mesh's AOBB

public int[][] getFaces()

Returns the list of faces

Returns

the list of faces; each face is a list of 3 integers which are indices for the list of vertices

public RefractionSim.Vector[] getVerts()

Returns the list of vertices

Returns

the list of vertices

public RefractionSim.Vector[] getNormals()

Returns the list of normals

Returns

a list of normals corresponding to the faces with the same subscripts

public double[] getDs()

Returns the list of values for d

Returns

a list of the d values corresponding to the faces with the same indices

private RefractionSim.Vector normal(

int[] face)
Calculates and returns the normalised normal to face; the normal is in the direction the face is 'facing', which is the direction from which the face's vertices are listed in clockwise order
Parameters
face - the face which the normal needs to be calculated for
Returns
the unit length normal to face

Fields

private faces

private verts

private normals

private ds

private boxVerts

public static final class **RefractionSim.Mesh.Primitive** extends java.lang.Enum

An enumerated type that specifies the shapes for which the Mesh class can generate geometry

Constructors

private Mesh.Primitive(
String shape)
Stores the user-friendly name of the shape
Parameters
shape - the string representation of the shape

Methods

public static RefractionSim.Mesh.Primitive[] values()

public static RefractionSim.Mesh.Primitive valueOf(
String name)

public java.lang.String toString()
Returns the user-friendly name of the shape
Returns
the string representation of the shape

Fields

public static final CUBE

public static final CUBOID

public static final TRIANGULAR_PRISM

public static final SPHERE

public static final CONVEX_LENS

public static final CONCAVE_LENS

public static final HALF_CYLINDER

private shape

public class **RefractionSim.Object3D**

General class for any object that exists in 3-D space such as the camera

Constructors

public Object3D(
Mesh mesh,

Color color)

Constructor for the Object3D class

Parameters

mesh - the geometry of the 3-D object (null if there isn't any)

color - the colour of the 3-D object (irrelevant if the mesh is null and may also be null in this case)

Methods

public void setID(
int newID)

Sets the value of ID to newID

Parameters

newID - the new value for ID

public int getID()

Returns ID, the index of the object in the viewport's objectList

Returns

the object's ID

public RefractionSim.Mesh getMesh()

Returns the geometry of the object; null if there is no geometry

Returns

the geometry of the object

public RefractionSim.Vector[] getBoxVerts()

Returns the a list of the vertices of the object's arbitrarily orientated bounding box (AOBB)

Returns

the vertices of the object's bounding box

public java.awt.Color getColor()

Returns the object's colour

Returns

the colour of the object

public void displace(
Vector displacement)

Moves the object in 3-D space by the displacement vector

Parameters

displacement - 3-row vector representing movement in the world's x, y and z directions

Throws

IllegalArgumentException - if displacement is not a 3-row vector

public RefractionSim.Vector getOrigin()

Returns the location of the object's origin which other points are measured relative to in object space

Returns

the location of the object's origin in world space

public void setOrigin(
Vector origin)

Moves the object to a new position

Parameters

origin - the new position of the object's origin in world space

Throws

IllegalArgumentException - if origin is not a 3-row vector

public void rotate(
Matrix rotation)

Rotates an object about its origin; the vertices' co-ordinates aren't changed, but the object's basis vectors (columns of the orientation matrix) are changed

Parameters

rotation - a 3 by 3 matrix representing the angular displacement of the new orientation from the old one

Throws

IllegalArgumentException - if rotation is not a 3 by 3 matrix; the matrix should also represent a rotation, although this will not throw an exception

public RefractionSim.Matrix getOrientation()
Returns the matrix representing the orientation of the object relative to world/upright space
Returns
the 3 by 3 matrix representing the object's orientation relative to world/upright space

public void setOrientation(
Matrix orientation)
Sets the orientation of the object to the
Parameters
orientation - a 3 by 3 matrix representing an orientation
Throws
IllegalArgumentException - if the matrix is not 3 by 3; it should also represent a rotation, although this will not throw an exception

public void orbit(
double heading,
double pitch)
Rotates the object about the origin of world space; the object's origin and rotation are affected. This method assumes the object to be facing the world's origin
Parameters
heading - the angle of rotation clockwise around the world's y-axis
pitch - the angle of rotation clockwise around the object's x-axis

Fields

protected ID

protected mesh

protected color

protected orientation

protected origin

protected boxVerts

public class **RefractionSim.Ray**

Class for rays; lines with a starting point and a direction

Constructors

public Ray(
Vector p,
Vector v)
Constructor for the Ray class
Parameters
p - the starting point of the ray
v - the direction of the ray

Methods

public RefractionSim.Vector getP()
Returns the starting point of the ray
Returns
the point where the ray starts

public RefractionSim.Vector getV()
Returns the direction the ray extends from its starting point
Returns
the direction of the ray

Fields

private p

private v

public class **RefractionSim.RayBox** extends RefractionSim.Object3D

Class for ray boxes

Constructors

public RayBox()

Constructor for the RayBox class that generates the ray box's geometry, sets its colour, position and orientation and creates the light beam with geometry

Methods

public void setOrigin(
Vector origin)

Sets the origin of both the ray box and light beam

Parameters

orgin - the new origin for the ray box and light beam

Throws

IllegalArgumentException - if origin is not a 3-row vector

public java.lang.String getLabel()

Returns the label of the ray box

Returns

the label of the ray box

public void setLabel(
String newLabel)

Sets the ray box's label to the newLabel parameter

Parameters

newLabel - the new label for the ray box

public RefractionSim.Beam getLightBeam()

Returns the light beam for the ray box

Returns

the light beam for the ray box

public boolean getAnglesVisible()

Returns whether angles are set to be visible for the ray box/light beam

Returns

whether or not angle visibility is on

public void setAnglesVisible(
boolean showAngles)

Sets the visibility of angles for the ray box/light beam

Parameters

showAngles - whether or not angles should be displayed for the ray box/light beam

public int getBeamThickness()

Returns the thickness of the ray box's light beam between 1 and 10 inclusive

Returns

the thickness of the light beam

public void setBeamThickness(
int newThickness)

Sets the radius of the light beam from a thickness value between 1 and 10 inclusive

Parameters

-

public boolean isLocalPitchInverted()

Returns whether or not the ray box is upside down, meaning that the effect of a change in the local pitch slider is negated

Returns

whether or not the local pitch slider is inverted for the ray box

public void toggleLocalPitchInverted()

Toggles whether or not the local pitch slider is inverted for the ray box

public void rotate(
Matrix rotation)

Rotates both the ray box and light beam about their origins by the angular displacement represented by the rotation matrix

Parameters

rotation - a 3 by 3 matrix representing the angular displacement of the new orientation from the old one

Throws

IllegalArgumentException - if rotation is not a 3 by 3 matrix; the matrix should also represent a rotation, although this will not throw an exception

public void rotate(
double heading,
double pitch)

Rotates both the ray box and light beam about their origins by the heading and pitch angles

Parameters

heading - the angle of rotation clockwise about the world's y-axis

pitch - the angle of rotation about the ray box's x-axis

public void orbitAboutOrigin(
double heading,
double pitch)

Rotates both the ray box and light beam about the world's origin by the heading and pitch angles. Unlike the orbit method of Object3D this doesn't assume that the ray box is facing the world's origin

Parameters

heading - the angle of rotation clockwise about the world's y-axis

pitch - the angle of rotation clockwise about the horizontal vector perpendicular to the vector from the origin to the ray box if the ray box had a y co-ordinate of 0

Fields

private label

private lightBeam

private localPitchInverted

public class **RefractionSim.RefractionSimulator** extends javax.swing.JFrame

Class for windows of the application also containing the public static main method which the system calls to start the application

Constructors

public RefractionSimulator()

Constructor for the RefractionSimulator class which sets properties for the window as well as generating the contents of the window and drawing it

Methods

public static void main(
String[] args)

Called by the system when the application is started. This creates an instance of RefractionSimulator and sets up the window

Parameters

args - the parameter passed by the system which isn't needed in this application

public void updatePropertiesPanel(
RayBox rayBox)

Regenerates the properties panel for rayBox as the selected ray box and replaces the old the old properties panel

Parameters

rayBox - the selected ray box which the properties panel will display information for. For an empty selection, null should be used as the parameter

public void updateMenuBar()

Regenerates the menu bar and replaces the old menu bar

Fields

private userInterface

private content

public class **RefractionSim.Target** extends RefractionSim.Object3D

Class for 3-D objects which act as a transmission medium for light

Constructors

public Target(
Mesh.Primitive shape,
Color color,
int materialID)

Constructor for the Target class

Parameters

shape - the shape of the target object from the selection of primitive geometries
color - the overall colour of the object including transparency
materialID - the index of the material of the object in the viewport's materials list

Methods

public int getMaterial()

Returns the index of the target's material in the viewport's list of materials

Returns

the index of the target's material

public void setMaterial(
int materialID)

Sets the material to that referenced by the materialID parameter

Parameters

materialID - the index of the material the target is to be changed to

public java.lang.String getShape()

Returns the name of the shape of the target

Returns

the name of the target's shape

Fields

private materialID

private shape

public class **RefractionSim.UIController**

Class for objects that generate the user interface

Constructors

public UIController(
RefractionSimulator window)

Constructor for the UIController class which generates the viewport, menu bar and properties panel

Parameters

window - the window which will contain this new viewport, menu bar and properties panel

Methods

private void createViewport()

Generates a viewport to fill all of the space in the window left by the properties panel and menu bar. The viewport can then be retrieved using getViewport().

public RefractionSim.Viewport getViewport()

Returns the generated viewport

Returns

the viewport that was generated by createViewport()

public void createMenuBar()

Generates a menu bar which can then be retrieved using getMenuBar()

private javax.swing.JMenu getFileMenu()
Generates and returns the File menu which is to be added to the menu bar
Returns
the File menu

private javax.swing.JMenu getAddMenu()
Generates and returns the Add menu which is to be added to the menu bar
Returns
the Add menu

private void addTargetWorldMenus(
JMenuBar menuBar)
Generates the Target and World menus and adds them to the menu bar. This method is more efficient than generating the menus separately because the target's Material menu and the World menu are similar
Parameters
menuBar - the menu bar which the menus are to be added to

private javax.swing.JMenu getViewMenu()
Generates and returns the View menu which is to be part of the menu bar
Returns
the View menu

private javax.swing.JMenu getCameraPositionsMenu()
Generates and returns the menu of preset camera positions/orientations which is to be a submenu of the View menu
Returns
the menu of camera positions/orientations

public javax.swing.JMenuBar getMenuBar()
Returns the last menu bar that was generated by createMenuBar()
Returns
the most recent menu bar that was generated by a call to createMenuBar()

public void buildPropertiesPanel(
RayBox rayBox)
Generates a properties panel that can be retrieved by getPropertiesPanel()
Parameters
rayBox - the selected ray box

private javax.swing.JPanel getLabelPanel(
RayBox rayBox)
Generates and returns the panel for setting the label of the selected ray box or deleting the selected ray box
Parameters
rayBox - the selected ray box
Returns
the panel containing a label, a text field and a delete button

private javax.swing.JPanel getDisplayAnglesPanel(
RayBox rayBox)
Generates and returns the panel containing the checkbox indicating whether angles are displayed for the selected ray box
Parameters
rayBox - the selected ray box
Returns
the display angles panel

private javax.swing.JPanel getBeamThicknessPanel(
RayBox rayBox)
Generates and returns the panel containing a beam thickness input field, label and increment and decrement buttons
Parameters
rayBox - the selected ray box
Returns
the panel containing components relating to the beam thickness setting of the selected ray box

private javax.swing.JSlider getHeadingSlider(
int parentHeight,
double heading)
Generates and returns a labelled vertical slider between -SLIDER_MAX (representing -pi radians) and SLIDER_MAX (representing pi radians) with the initial value as heading converted from radians. Labels are in the angular units specified by the user (degrees or radians) and ticks indicate increments on the slider
Parameters
parentHeight - the height in pixels of the component that is to contain the slider
heading - the global or local heading angle of the selected ray box in radians
Returns
the slider representing a local or global heading of the selected ray box

private javax.swing.JSlider getPitchSlider(
int parentHeight,
double pitch)
Generates and returns a labelled vertical slider between -SLIDER_MAX (representing -pi/2 radians) and SLIDER_MAX (representing pi/2 radians) with the initial value as pitch converted from radians. Labels are in the angular units specified by the user (degrees or radians) and ticks indicate increments on the slider
Parameters
parentHeight - the height in pixels of the component that is to contain the slider
pitch - the global or local pitch angle of the selected ray box in radians
Returns
the slider representing a local or global pitch of the selected ray box

public javax.swing.JPanel getPropertiesPanel()
Returns the properties panel generated by the last call to buildPropertiesPanel
Returns
the properties panel generated by the last call to buildPropertiesPanel

Fields

private viewport

private menuBar

private propertiesPanel

private fullWidth

private fullHeight

private static final PROPS_PANEL_WIDTH

private static final SLIDER_MAX

private class **RefractionSim.UIController.FileSaver** extends javax.swing.JFileChooser

Class for dialog boxes which allow the user to save an image

Constructors

public UIController.FileSaver()
Constructor for the FileSaver class which sets the allowable file formats and sets gif as the default

Methods

public void approveSelection()
Called when the user clicks 'Save' to open a dialog box if a file already exists with this name in this directory

public java.io.File getFileWithExtension()
Returns a file (with the file extension included in its path) into which image data can be written
Returns
a file (with the file extension included in its path) into which image data can be

written

public java.lang.String extensionFromFileName(
String fileName)
Returns the string of characters after the last "." in fileName in lower case; if there is no "." in fileName then "" will be returned
Parameters
fileName - the name or path of a file
Returns
the file extension of the file name (characters after the last "." in lower case) or an empty string

Fields **private gifDescription**

private pngDescription

private jpgDescription

private class **RefractionSim.UIController.IncDecActionListener** implements java.awt.event.ActionListener

Class for ActionListeners of buttons that increment or decrement the value in a JTextField by 1

Constructors **public UIController.IncDecActionListener(**
JTextField textField)
Constructor for the IncDecActionListener class
Parameters
textField - the text field that an ActionEvent increments or decrements the value of

Methods **public void actionPerformed(**
ActionEvent event)
Called when the button is clicked and increments or decrements the value in textField by 1
Parameters
event - contains details of the action that triggered this event

Fields **private textField**

private class **RefractionSim.UIController.TextFieldFocusListener** implements java.awt.event.FocusListener

Class for FocusListeners of text fields that need to trigger an ActionEvent when they lose or gain focus

Constructors **public UIController.TextFieldFocusListener(**
ActionListener actionListener)
Constructor for the TextFieldFocusListener class
Parameters
actionListener - the ActionListener of the text field

Methods **public void focusGained(**
FocusEvent event)
Called when the text field gains focus to trigger an ActionEvent
Parameters
event - contains details of the action that triggered this event

public void focusLost(
FocusEvent event)
Called when the text field loses focus to trigger an ActionEvent
Parameters
event - contains details of the action that triggered this event

Fields **private actionListener**

private class **RefractionSim.UIController.BeamThicknessActionListener** implements java.awt.event.ActionListener

Class for ActionListeners of beam thickness text fields that validates the data entered, rounding to an appropriate value or reverting to the last appropriate value

Constructors **public UIController.BeamThicknessActionListener(**
String initialString)
Constructor for the BeamThicknessActionListener class
Parameters
initialString - the initial string in the text field

Methods **public void actionPerformed(**
ActionEvent event)
Called when the text field gains or loses focus or the user hits enter with it in focus
Parameters
event - contains details of the action that triggered this event

Fields **private lastValid**

private class **RefractionSim.UIController.MaterialNameActionListener** implements java.awt.event.ActionListener

Class for ActionListeners of material name input fields that restricts the name to 30 characters with no whitespace at the beginning or end (blank string replaced with)

Constructors **private UIController.MaterialNameActionListener()**

Methods **public void actionPerformed(**
ActionEvent event)
Called when the text field loses or gains focus or the user hits enter with it in focus
Parameters
event - contains details of the action that triggered this event

private class **RefractionSim.UIController.RefractiveIndexActionListener** implements java.awt.event.ActionListener

Class for ActionListeners of refractive index input fields that restricts values between 1.0 and 100.0 inclusive and reverts non-numerical input to the last valid value

Constructors **public UIController.RefractiveIndexActionListener(**
String initialString)
Constructor for the RefractiveIndexActionListener class
Parameters
initialString - the initial contents of the text field

Methods **public void actionPerformed(**
ActionEvent event)
Called when the text field loses or gains focus or the user hits enter with it in focus
Parameters
event - contains details of the action that triggered this event

Fields **private lastValid**

private class **RefractionSim.UIController.SliderListener** implements javax.swing.event.ChangeListener

Class for ChangeListeners of sliders that apply the relevant transformation to the selected ray box while the slider is being dragged

Constructors

public UIController.SliderListener(
double previousValue)

A constructor for SliderListeners that apply to sliders which don't have any affect on other sliders. These are the local orientation sliders

Parameters

previousValue - the initial value of the slider converted to radians

public UIController.SliderListener(
double previousValue,
JSlider affectedSlider)

A constructor for a SliderListener that applies to a slider which affects one other slider (the global heading slider affects the local heading)

Parameters

previousValue - the initial value of the slider this object is to be a listener for in radians

affectedSlider - the slider which is affected by changes to the other slider

public UIController.SliderListener(
double previousValue,
JSlider directlyAffectedSlider,
JSlider indirectlyAffectedSlider,
RayBox rayBox)

A constructor for a SliderListener that applies to a slider which affects one other slider and occasionally a second other slider (the global pitch slider directly and sometimes needs to change the local heading)

Parameters

previousValue - the initial value of the slider this object is to be a listener for in radians

directlyAffectedSlider - the slider which is always affected by changes to the slider being listened for

indirectlyAffectedSlider - the slider that is sometimes affected by changes to the slider being listened for

rayBox - the selected ray box

Methods

public void stateChanged(
ChangeEvent event)

Called when the slider is dragged (including while it is still being dragged) or the value changed by another slider that affects it

Parameters

event - contains details of the action that triggered this event

private double headingFromSliderVal(
int sliderValue)

Returns the change in heading in radians from the last time the slider was changed (or from when the slider was generated) and updates prevValue

Parameters

sliderValue - the new value of the slider in non-standard units

Returns

the change in heading in radians

private double pitchFromSliderVal(
int sliderValue)

Returns the change in pitch in radians from the last time the slider was changed (or from when the slider was generated) and updates prevValue

Parameters

sliderValue - the new value of the slider in non-standard units

Returns

the change in pitch in radians

private void updateParallelSlider(
int newSliderValue)

For a global slider, this method changes the sliders it affects

Parameters

newSliderValue - the new value of the slider this object is a listener for in non-standard units

private void invertPitchSlider()

For a global pitch slider, this method inverts the local pitch slider by changing its name, toggling a ray box property and adding or subtracting pi radians from the local heading

Fields

private prevValue

private parallelSlider

private secondaryParallelSlider

private rayBox

public class **RefractionSim.Vector**

Class for vectors of all kinds. All vectors are column vectors because matrices are column-major.

Constructors

public Vector(
int n)

Constructor for the Vector class which sets all elements to zero

Parameters

n - number of elements that the new vector is to have

Throws

IllegalArgumentException - if n is less than or equal to 2 (as this would produce nothing or a single value)

Methods

public int getN()

Returns the number of rows the matrix has

Returns

the number of rows the matrix has

public void setElement(
int i,
double newValue)

Sets the value of a single element in the vector to that of the newValue parameter

Parameters

i - the index (row) of the element to change (indices start at 0)
newValue - the value which the specified element should be changed to

Throws

ArrayIndexOutOfBoundsException - if i >= number of elements (rows)

public void setElements(
double[] newValues)

Sets all elements of the vector to the values in the newValues array

Parameters

newValues - the array of new values that the vector elements should be set to

Throws

IllegalArgumentException - if the length of the array is not equal to the number of elements in the vector

public void setElements(
Vector newValues)

Copies the values of the elements of the vector newValues to the corresponding elements in the vector for which this method is being called (the two vectors then don't reference the same locations)

Parameters

newValues - the vector of new values the vector elements should be set to

Throws

IllegalArgumentException - if the newValues parameter is not the same length as the vector for which this method is called

public double getElement(
int i)

Gets the value of a single element in the vector

Parameters

i - the index (row) of the element to return the value of

Returns

the value of the specified element

Throws

ArrayIndexOutOfBoundsException - if i >= number of elements or i < 0

public double[] getElements()

Returns the values of all elements in an array

Returns

the array of values in the vector

public RefractionSim.Vector add(
Vector toAdd)

Returns the sum of this vector and the toAdd parameter vector

Parameters

toAdd - the vector to add to the vector for which this method is being called

Returns

the sum of this vector and the vector passed as a parameter (the return vector will be the same size as both vectors being added)

Throws

IllegalArgumentException - if the toAdd vector is not the same size as the vector for which this method is being called

public RefractionSim.Vector subtract(
Vector toSubtract)

Returns a - b where a is the vector for which this method is called and b is the toSubtract parameter vector

Parameters

toSubtract - vector to subtract from the vector for which this method is called

Returns

a - b where a is the vector for which this method is called and b is the toSubtract parameter vector

Throws

IllegalArgumentException - if the two vectors don't have the same dimensions

public double dotProduct(
Vector toDot)

Returns the dot product of the vector for which this method is called and the toDot parameter vector. The dot product is associative, meaning that a.b = b.a

Parameters

toDot - the vector to dot with

Returns

the dot product of the two vectors (representing |a||b|cos(x) where a and b are the two vectors and x is the angle between them - this is the same as the component of a parallel to b multiplied by |a|)

public RefractionSim.Vector scale(
double scaleFactor)

Returns the vector multiplied by a scalar value

Parameters

scaleFactor - the scalar value by which the vector is to be multiplied

Returns

the vector produced when the original vector is multiplied by scaleFactor

public RefractionSim.Vector crossProduct(

Vector toCross)

Returns a x b (a crossed with b) where a is the vector for which this method is being called and b is the toCross parameter vector. Both a and b must have 3 elements (rows) and the cross product will also have 3. The cross product represents a vector perpendicular to both vectors with modulus |a||b|sin(x) where x is the angle between the vectors a and b

Parameters

toCross - the 3-row vector b in a x b where a is the vector for which this method is being called

Returns

a vector with 3 elements (rows) which represents a x b (a crossed with b) a is the vector for which this method is being called and b is the toCross parameter vector

public double modulus()

Returns the modulus (a.k.a. length, euclidean norm or 2-norm) of the vector

Returns

the modulus/length/euclidean norm/2-norm of the vector

public boolean isNormalised()

Returns true if the vector is normalised (has modulus 1). Some leeway is acceptable for the vector to be considered normalised

Returns

whether or not the vector is normalised

public RefractionSim.Vector normalise()

Returns the normalised version of the original vector (meaning it has modulus 1 with slight leeway)

Returns

the normalised version of the original vector

Fields

private n

private vec

private modulus

private modulusKnown

public class **RefractionSim.Viewport** extends javax.swing.JPanel

Class for 3-D viewports which deals with the user interface requirements of being a subclass of JPanel as well as handling its own 'scene' of objects

Constructors

public Viewport(
int frameX,
int frameY)

Constructor for the Viewport class which sets its size, prepares it for rendering, adds the camera and target to the scene and sets a listener for all events within the viewport that need handling

Parameters

frameX - width of the viewport in pixels
frameY - height of the viewport in pixels

Methods

private void initialiseMaterials()

Defines some common materials by giving them names and refractive indices

private void initialiseScene()

Sets up the default camera and target and adds them to the scene/objectList; objectList[0] is always the camera and objectList[1] is always the target

public void paintComponent(
Graphics g)

Redraws the contents of the viewport; the scene is re-rendered into the buffers including the outline for the selected ray box, then the buffers are drawn inside the viewport component and ray box labels and angles drawn on top. This method should be called via repaint() which decides whether it is worthwhile drawing the component and passes an appropriate parameter.

private void render()
Clears the buffers and renders the 3-D scene to the buffers from the camera's point of view

private void outlineSelectedObj()
Overwrites areas of the frame buffer in order to create a bright orange outline 1 pixel thick around the selected object so that the user can identify which object is selected

private void writeAngles(
Graphics g,
FontRenderContext frc)
Draws/Writes all angles of incidence, refraction and reflection inside the viewport (on top of anything already drawn in the viewport)
Parameters
g - the graphics context for the viewport which allows text to be drawn in the viewport
frc - the FontRenderContext of the 2-D graphics context (which should have anti-aliasing) which allows the width of text to be determined without drawing

private void writeRayBoxLabels(
Graphics g,
FontRenderContext frc)
Draws/Writes all ray box labels inside the viewport (on top of anything that has already been drawn)
Parameters
g - the graphics context for the viewport which allows text to be drawn in the viewport
frc - the FontRenderContext of the 2-D graphics context (which should have anti-aliasing) which allows the ascent (maximum height above the baseline) of text to be determined without drawing

public static RefractionSim.Object3D[] getObjectList()
Returns the list of 3-D objects in the scene where the item at index 0 is the camera and the item at index 1 is the target
Returns
the list of all 3-D objects in the scene

private boolean inView(
Object3D obj)
Returns true if the bounding box of an object is at least partially visible to the camera (but the object itself may still be completely out of sight)
Parameters
obj - the 3-D object to be checked for visibility
Returns
whether or not the bounding box of obj is in view of the camera

private RefractionSim.Vector project(
Vector cameraCoord)
Maps a point from camera space to normalised clip space and returns the point as a Vector object
Parameters
cameraCoord - the point in camera space which is to be mapped to normalised clip space
Returns
cameraCoord mapped to normalised clip space
Throws
IllegalArgumentException - if the cameraCoord parameter is not a 3-row vector

private java.awt.Color calcFaceColor(
Vector p0,
Vector p1,
Vector p2,
Object3D object)
Returns the colour to render a particular face in based on the object's overall colour and how

much the face is pointing towards the camera in normalised clip space

Parameters

p0 - the first vertex of the face in normalised clip space. It is important that the order of the vertices is correct

p1 - the second vertex of the face

p2 - the third vertex of the face

object - the object to which the face belongs

Returns

the colour to render the face defined by the three input points

Throws

IllegalArgumentException - if any of p0, p1 and p2 is not a 3-row vector

public void addRayBox(
RayBox newRayBox)

Adds the parameter ray box and its beam to the scene and makes the ray box the selected object before updating the user interface and viewport. A dialog box informs the user if they have too many ray boxes to add any more

Parameters

newRayBox - the ray box to add to the scene along with its beam

public void removeRayBox()

Removes the selected ray box and its beam from the scene and updates the user interface and viewport; there is no longer a selected object

Throws

IllegalArgumentException - if the selected object isn't a ray box

private void clearBuffers()

Resets the frame buffer, depth buffer and object buffer in preparation for re-rendering

public int getWidth()

Returns the width of the viewport in pixels

Returns

the width of the viewport in pixels

public int getHeight()

Returns the height of the viewport in pixels

Returns

the height of the viewport in pixels

public static java.lang.String[] getMaterials()

Returns an array of the names of all the materials - preset and custom - in the same order as the refractive indices array

Returns

an array of the names of all preset and custom materials

public static double[] getRefractiveIndices()

Returns an array of the absolute refractive indices of all the preset and custom materials in the same order as the material names array.

Returns

the refractive indices of all the materials

public static int getWorldMaterial()

Returns the index of the material that the world is set to

Returns

the index of the material that the world is set to

public void setWorldMaterial(
int materialID)

Sets the material of the world to that with the index passed as a parameter

Parameters

materialID - the index of the material that the world should be

Throws

IllegalArgumentException - if there is currently not a material with the index materialID

public int getTargetMaterial()

Returns the index of the material that the target is set to

Returns

the index of the material that the target is set to

public void setTargetMaterial(
int materialID)

Sets the material of the target to that with the index passed as a parameter

Parameters

materialID - the index of the material that the target should be set to

Throws

IllegalArgumentException - if there is currently not a material with the index materialID

public static int getNumOfMaterials()

Returns the total number of materials that are defined

Returns

the total number of materials

public void addMaterial(
String materialName,
double refractiveIndex)

Creates a new material with the properties specified by the parameters and adds it to the end of the list of materials (which is a combination of the list of material names and the list of refractive indices)

Parameters

materialName - the name of the new material
refractiveIndex - the absolute refractive index of the new material

public java.lang.String getTargetShape()

Returns the name of the current shape of the target material

Returns

the name of the current shape of the target material

public void setTargetShape(
Mesh.Primitive newShape)

Sets the geometry of the target to that defined by newShape and recalculates the paths of all beams in the scene

Parameters

newShape - the new shape of the target material

public void recalculateBeams()

Recalculates the paths of all beams in the scene but does not re-render

public boolean isOrthographic()

Returns true if orthographic projection is currently being used

Returns

whether orthographic projection is being used

public void toggleOrthographic()

Switches from orthographic projection to perspective projection or from perspective to orthographic and re-renders

public boolean areAnglesInDegrees()

Returns true if angles are set to display in degrees rather than radians

Returns

whether angles are displayed in radians

public void toggleAngleUnits()

Changes angles from degrees to radians or vice versa and updates the interface accordingly

public void globallyRotateRayBox(
double heading,
double pitch)

Orbits the selected ray box about the world origin by the heading and pitch specified

Parameters

heading - the angular displacement about the world's y-axis
pitch - the angular displacement about the world's x-axis after rotation by the heading

public void locallyRotateRayBox(
double heading,
double pitch)

Rotates the selected ray box about its origin by the heading and pitch specified

Parameters

heading - the angular displacement about the world's y-axis

pitch - the angular displacement about the world's x-axis after rotation by the heading

private void calcClipMatrix()

Sets up clipMatrix to map camera space to clip space (when using perspective projection) for the given zoomX, zoomdY, NEAR_CLIP and FAR_CLIP

private void rasterise(
Vector point0,
Vector point1,
Vector point2,
Vector normal,
double d,
Color faceColor,
int objectID)

From information in extended screen space (screen space with depth), draws the visible parts of a face into the buffers

Parameters

point0 - the first vertex (index 0) of the face in screen space

point1 - the second vertex (index 1) of the face in screen space

point2 - the third vertex (index 2) of the face in screen space

normal - the normalised normal to the face in extended screen space

d - the value in the expression p.n = d where p is a point on the face and n is the normalised normal to the face (all in extended screen space)

faceColor - the colour to render the face

objectID - the ID of the object to which this face belongs (the index of the object in objectList)

private void rasteriseHalfFace(
int initialY,
int finalY,
double xShort,
double xTall,
double dxShort,
double dxTall,
double minDepth,
Vector normal,
double d,
Color faceColor,
int objectID)

Sets pixels in the buffers where the face is visible for initialY <= y < finalY

Parameters

initialY - the first row of pixels (lowest y value)

finalY - the row of pixels after the last (highest y value)

xShort - the x co-ordinate of the shorter edge when the y co-ordinate is initialY

xTall - the x co-ordinate of the taller edge when the y co-ordinate is initailY

dxShort - the change in x of the shorter edge when y is increased by 1

dxTall - the change in x of the taller edge when y is increased by 1

minDepth - the lowest depth value of any point on the face in extended screen space

normal - the normalised normal to the face in extended screen space

d - the value in the expression p.n = d where p is a point on the face and n is the normalised normal to the face (all in extended screen space)

faceColor - the colour to render the face

objectID - the ID of the object to which the face belongs (the index of the object in objectList)

private void rasteriseFaceRow(
int startX,
int endX,
int pixelY,

double minDepth,
Vector normal,
double d,
Color faceColor,
int objectID)

Sets pixels in the buffers where the face is visible for startX <= x < endX and y co-ordinate pixelY

Parameters

startX - the x co-ordinate of the first pixel on this row contained by the triangle

endX - the x co-ordinate of the pixel after the last on this row contained by the triangle

pixelY - the y co-ordinate of the row of pixels being set

minDepth - the lowest depth value of any point on the face in extended screen space

normal - the normalised normal to the face in extended screen space

d - the value in the expression p.n = d where p is a point on the face and n is the normalised normal to the face (all in extended screen space)

faceColor - the colour to render the face

objectID - the ID of the object to which the face belongs (the index of the object in objectList)

private void setPixel(
int x,
int y,
double depth,
Color color,
int objectID)

Changes the frame buffer, depth buffer and object buffer for a single pixel

Parameters

x - the x co-ordinate of the pixel to change

y - the y co-ordinate of the pixel to change

depth - the depth of the current face at the centre of this pixel

color - the rendered colour of the face (including alpha)

objectID - the ID of the object this face belongs to (the index of the object in objectList)

public void orbit(
int xChange,
int yChange)

Orbit the camera around the world's origin where xChange is directly proportional to the heading and yChange directly proportional to the pitch

Parameters

xChange - the signed change in the x position of the user's cursor between dragging events

yChange - the signed change in the y position of the user's cursor between dragging events

public void zoom(
double scrollAmount)

Moves the camera towards or away from the world's origin

Parameters

scrollAmount - the signed number of notches the mouse wheel is moved down or the equivalent amount dragged down with the right mouse button held

public void click(
int x,
int y)

Changes the selection to the ray box visible at position (x, y) in the viewport (otherwise the selection becomes empty) and updates the properties panel

Parameters

x - the x co-ordinate of the pixel the user clicked relative to the top left of the viewport

y - the y co-ordinate of the pixel the user clicked relative to the top left of the viewport

public void updateLabel(
String newLabel)

Sets the label of the selected object to newLabel

Parameters

newLabel - the string to set the label of the selected object to

public void updateBeamThickness(
int newThickness)

Sets the thickness of the selected ray box's beam to newThickness

Parameters

newThickness - the relative value that the thickness of the selected ray box's beam should be set to

public void toggleShowAngles()

Changes angle visibility from on to off or off to on for the selected ray box and refreshes the viewport including angles

public void setViewFront()

Positions and orientates the camera to face directly forwards at the same distance from the origin as before

public void setViewBack()

Positions and orientates the camera to face directly backwards at the same distance from the origin as before

public void setViewLeft()

Positions and orientates the camera to face directly right at the same distance from the origin as before

public void setViewRight()

Positions and orientates the camera to face directly left at the same distance from the origin as before

public void setViewTop()

Positions and orientates the camera to face directly down at the same distance from the origin as before

public void setViewBottom()

Positions and orientates the camera to face directly up at the same distance from the origin as before

public void saveImage(
File outputFile,
String fileExtension)

Saves the contents of the viewport to a bitmapped image file with path outputFile and of type expressed by fileExtension

Parameters

outputFile - the path (including name with file extension) to save the image under
fileExtension - one of "gif", "jpg" and "png" which indicates the format for the file

Fields

private frameWidth

private frameHeight

private frameBuffer

private depthBuffer

private objectBuffer

private bgColor

private static orthographic

private zoomX

private zoomY

private static final NEAR_CLIP

private static final FAR_CLIP

private clipMatrix

private static objectList

private static objectListLength

private static worldMaterial

private selectedObjID

private anglesInDegrees

private static materials

private static refractiveIndices

private static numOfMaterials

public class **RefractionSim.ViewportListener** implements java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.awt.event.MouseWheelListener, java.awt.event.KeyListener

Class for all-purpose viewport listeners handling both mouse and keyboard events

Constructors **public ViewportListener()**

Methods

public void mousePressed(
MouseEvent event)
Called when any of the mouse buttons are depressed in the viewport to give the viewport focus and prepare for dragging

public void mouseDragged(
MouseEvent event)
Called when the user drags after depressing any of the mouse buttons in the viewport to orbit the camera around the world's origin or zoom if the right mouse button is used

public void mouseReleased(
MouseEvent event)
Called when any mouse button is released to end a drag

public void mouseClicked(
MouseEvent event)
Called when a mouse button is released without having moved since it was depressed to change the selection if the left mouse button was used

public void mouseWheelMoved(
MouseWheelEvent event)
Called when the user scrolls with the mouse wheel (middle mouse button) to move the camera closer to or further away from the world's origin (zooming)

public void keyReleased(
KeyEvent event)
Called when a key on the keyboard is released or held down to set the camera position and orientation or toggle between orthographic and perspective projection

public void keyPressed(
KeyEvent event)
Called when a key is pressed; no action is taken

public void keyTyped(
KeyEvent event)

Called when a key is pressed or held such that a character would be typed if a text field was in focus

public void mouseEntered(
MouseEvent event)
Called when the cursor enters the viewport

public void mouseExited(
MouseEvent event)
Called when the mouse exits the viewport

public void mouseMoved(
MouseEvent event)
Called when the mouse is moved inside the viewport

Fields **private dragging**

private prevX

private prevY