Standard Draw 3D

Complete Reference Manual

Hayk Martirosyan

Introduction

Standard Draw 3D is a Java library with the express goal of making it simple to create three-dimensional models, simulations, and games. If you have not used StdDraw3D before, make sure to go through the basic tutorial before coming here.

This document is the official reference manual for Standard Draw 3D. Listed below is every method in the library, categorized by functionality. Most of the library is in the form of static methods, but there are also nested classes that are crucial to the way StdDraw3D works.

Table of Contents

Each section of the table of contents is hyperlinked to the corresponding section of the manual.

- Cone
- Cylinder
- Ellipsoid

Advanced 3D Shapes

- 3D Text
- Tubes
- External Model

Display and Animation

- Showing
- Clearing

Static Camera Placement

- Camera Position and Orientation
- Camera Direction

Interactive Navigation

- Mode Descriptions
- Setting the Camera Mode
- Orbit Center

Points, Lines, and Surfaces

- Points
- Lines
- Polygons
- Triangles

Saving and Loading

- Image File
- 3D Scene

Lights

- Light Utilities
- Ambient Light
- Point Light
- Directional Light
- Fog

Sounds

- Sound Utilities
- Ambient Sound
- Point Sound

Mouse and Keyboard

- Mouse Buttons
- Mouse Position
- Key Pressed
- Key Stack

2D Overlay

- Circle

- Square
- Rectangle
- Ellipse
- Pixel
- Point
- Line
- Arc
- Polygon
- Text
- Picture

Random Generators

- Color
- Vector Direction

Shape Class

Static Shape Manipulation

- Combine
- Copy

Position

- Move Absolute
- Move Relative
- Set Position

Orientation

- Rotate Absolute
- Rotate Relative
- Set Orientation
- Rotate about Axis
- Look At
- Set Direction

Size

Color

Matching

Hiding

Camera Class

Getting the Camera Object

Methods from Shape Class

Additional Methods

- Pair
- Rotate FPS

Light Class

Methods from Shape Class

Power of a Point Light

Vector3D Class

Fields

Constructors

Methods

- Addition
- Subtraction
- Multiplication
- Magnitude
- Direction
- Distance
- Dot Product
- Cross Product
- Angle
- Projection
- Reflection
- To String
- Draw

STATIC METHODS

These static functions are called by prefixing the method name with "*StdDraw3D.*". For example, to draw a sphere, call *StdDraw3D.sphere(x, y, z, r)*. Nested classes of StdDraw3D are documented below this section.

General Settings

Global settings of StdDraw3D are included here, and control various aspects of the drawing window and rendering process. Canvas size and scale should be defined at the start of each project.

Canvas Size

Sets the drawing window size to (w, h) pixels. Can also set the drawing window to fullscreen, which makes it as large as possible to fit the displaying monitor.

All Arguments

Type	Name	Description
double	**w**	width
double	**h**	height

Methods

Return Type	Name	Arguments
void	**setCanvasSize**	(w, h)
void	**fullscreen**	()

Scale

The "scale" of the drawing window is important to the way StdDraw3D works. In general, it is the coordinate range which you are likely to be drawing in (for all three dimensions). In your drawing window, is by default placed such that you can see the working volume specified by the *setScale()* function. Of course you can move the camera as desired, but navigation speed is also calibrated based on the scale, so it is useful to set a good scale. Calling *setScale()* without arguments reverts to the default scale of (0, 1).

Arguments

Type	Name	Description
double	**minimum**	the minimum x, y, and z coordinate of your working volume
double	**maximum**	the maximum x, y, and z coordinate of your working volume

Methods

Return Type	Name	Arguments
void	**setScale**	(minimum, maximum)
void	**setScale**	()

Perspective

The perspective of a 3D scene is the way it is projected onto a 2D computer monitor to create the appearance of three-dimensional depth. Objects that are closer appear larger in real life, and this is how they are drawn in StdDraw3D. This is called perspective projection, and the amount to which closer objects appear larger is proportional to the field of view. You can specify the field of view with the *setPerspectiveProjection()* function, or revert to the default of 0.9. A large field of view looks like a fisheye, while a small field of view is like looking through a telescope. Reasonable values of fov range from 0.5 to 3.0. Zero perspective can be set with *setParallelProjection()*, and is how all CAD programs display 3D objects.

Arguments

Type	**Name**	Description
double	**fov**	field of view of perspective projection

Methods

Return Type	**Name**	Arguments
void	**setPerspectiveProjection**	(fov)
void	**setPerspectiveProjection**	()
void	**setParallelProjection**	()

Mesh Control

Sets the number of trianglular divisions of curved shapes like spheres and cylinders. The default is 100 divisons. Decrease this number to increase performance. Can also get the current value.

Arguments

Type	**Name**	Description
int	**N**	number of triangular divisions

Methods

Return Type	**Name**	Arguments
void	**setNumDivisions**	(N)
int	**getNumDivisions**	()

Info Display

Toggles the information display in the upper-left corner.

Arguments

Type	**Name**	Description
boolean	**enabled**	state of the info display

Methods

Return Type	**Name**	Arguments
void	**setInfoDisplay**	(enabled)

Anti-Aliasing

Toggles simple anti-aliasing. Anti-aliasing makes graphics look much smoother, but is very resource heavy. It is good for saving images that look professional. The default is off. Currently not supported for OS-X, and only works with some PCs. Can also get the current state.

Arguments

Type	**Name**	Description
boolean	**enabled**	state of anti-aliasing

Methods

Return Type	**Name**	Arguments
void	**setAntiAliasing**	(enabled)
boolean	**getAntiAliasing**	()

Pen Settings

These functions change the propreties of the drawing pen, which influences the properties of things you draw. For example, a drawn sphere will be the pen color, a drawn line will have the pen radius, and drawn text will have the pen font.

Pen Color

Sets the pen to the given color. Can specify either a color, a color and a transparency, RGB components, or nothing (reverts to the default pen color). Can also get the current pen color.

All Arguments

Type	**Name**	Description
Color	**col**	desired color
int	**alpha**	optional transparency of **color** (0 – 255)
int	**r**	red component of an RGB color
int	**g**	green component of an RGB color
int	**b**	blue component of an RGB color

Methods

Return Type	**Name**	Arguments
void	**setPenColor**	(col)
void	**setPenColor**	(col, alpha)
void	**setPenColor**	(r, g, b)
void	**setPenColor**	()
Color	**getPenColor**	()

Pen Radius

Sets the pen stroke to the given radius, or reverts to the default of 0.002 if no arguments are given. Can also get the current radius.

Arguments

Type	**Name**	Description
double	**r**	desired pen radius

Methods

Return Type	Name	Arguments
void	**setPenRadius**	(r)
void	**setPenRadius**	()
float	**getPenRadius**	()

Pen Font

Sets the pen to the given Font object, or reverts to the default if no arguments are given. Can also get the current font.

All Arguments

Type	Name	Description
Font	**f**	desired pen font

Methods

Return Type	Name	Arguments
void	**setFont**	(f)
void	**setFont**	()
Font	**getFont**	()

Background

The background of the drawing window is by black by default, but can be easily changed as needed. There are three options for specifying a background: you can set a solid color, a background image, or an image wrapped around a 3D background sphere.

Solid Color

Sets the background to a solid color.

Arguments

Type	Name	Description
Color	**color**	desired color

Methods

Return Type	Name	Arguments
void	**setBackground**	(color)

Image

Sets the background to a specified image, which is scaled to fit the window.

Arguments

Type	Name	Description
String	**imageURL**	filename or URL of desired background image

Methods

Return Type	Name	Arguments

```
void        setBackground (imageURL)
```

Spherical Panorama

Sets the background to a specified image, wrapped around as a spherical skybox.

Arguments

Type	Name	Description
String	**imageURL**	filename or URL of desired background image

Methods

Return Type	Name	Arguments
void	**setBackgroundSphere**	(imageURL)

Basic 3D Shapes

There are six basic three-dimensional shapes in StdDraw3D: spheres, cubes, boxes, cones, cylinders, and ellipsoids. Each primitive shape has a set of five drawing functions, depending on whether you want to specify rotation, texture, or wireframe properties. Sphere and cube are specific cases of ellipsoid and box, but are included for convenience. Every drawing function returns a Shape object, which can be moved, rotated, and scaled dynamically using the methods of the Shape class.

Sphere

Draws a sphere at (x, y, z) with radius r. Can specify Euler rotation angles (xA, yA, zA). Can be solid, wireframe, or textured.

All Arguments

Type	Name	Description
double	**x**	x-coordinate of the center
double	**y**	y-coordinate of the center
double	**z**	z-coordinate of the center
double	**r**	radius
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation
String	**imageURL**	filename or URL of wraparound texture

Return Value

Returns the drawn Shape object, which can be further manipulated.

Solid

Return Type	Name	Arguments
Shape	**sphere**	(x, y, z, r)
Shape	**sphere**	(x, y, z, r, xA, yA, zA)

Wireframe

Return Type	**Name**	Arguments
Shape	**wireSphere**	(x, y, z, r)
Shape	**wireSphere**	(x, y, z, r, xA, yA, zA)

Textured

Return Type	**Name**	Arguments
Shape	**sphere**	(x, y, z, r, xA, yA, zA, imageURL)

Cube

Draws a cube at (x, y, z) with radius r. Can specify Euler rotation angles (xA, yA, zA). Can be solid, wireframe, or textured.

All Arguments

Type	**Name**	Description
double	**x**	x-coordinate of the center
double	**y**	y-coordinate of the center
double	**z**	z-coordinate of the center
double	**r**	radius (half of side length)
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation
String	**imageURL**	filename or URL of wraparound texture

Return Value

Returns the drawn Shape object, which can be further manipulated.

Solid

Return Type	**Name**	Arguments
Shape	**cube**	(x, y, z, r)
Shape	**cube**	(x, y, z, r, xA, yA, zA)

Wireframe

Return Type	**Name**	Arguments
Shape	**wireCube**	(x, y, z, r)
Shape	**wireCube**	(x, y, z, r, xA, yA, zA)

Textured

Return Type	**Name**	Arguments
Shape	**cube**	(x, y, z, r, xA, yA, zA, imageURL)

Box

Draws a box at (x, y, z) with dimensions (w, h, d). Can specify Euler rotation angles (xA, yA, zA). Can be solid, wireframe, or textured.

All Arguments

Type	**Name**	Description
double	**x**	x-coordinate of the center
double	**y**	y-coordinate of the center

double	**z**	z – coordinate of the center
double	**w**	width
double	h	height
double	d	depth
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation
String	**imageURL**	filename or URL of wraparound texture

Return Value

Returns the drawn Shape object, which can be further manipulated.

Solid

Return Type	**Name**	Arguments
Shape	**box**	(x, y, z, w, h, d)
Shape	**box**	(x, y, z, w, h, d, xA, yA, zA)

Wireframe

Return Type	**Name**	Arguments
Shape	**wireBox**	(x, y, z, w, h, d)
Shape	**wireBox**	(x, y, z, w, h, d, xA, yA, zA)

Textured

Return Type	**Name**	Arguments
Shape	**box**	(x, y, z, w, h, d, xA, yA, zA, imageURL)

Cone

Draws a cone at (x,y,z) with radius r and height h. Can specify Euler rotation angles (xA, yA, zA). Can be solid, wireframe, or textured.

All Arguments

Type	**Name**	Description
double	**x**	x – coordinate of the center
double	**y**	y – coordinate of the center
double	**z**	z – coordinate of the center
double	**r**	radius
double	h	height
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation
String	**imageURL**	filename or URL of wraparound texture

Return Value

Returns the drawn Shape object, which can be further manipulated.

Solid

Return Type	**Name**	Arguments
Shape	**cone**	(x, y, z, r, h)
Shape	**cone**	(x, y, z, r, h, xA, yA, zA)

Wireframe

Return Type	**Name**	Arguments
Shape	**wireCone**	(x, y, z, r, h)
Shape	**wireCone**	(x, y, z, r, h, xA, yA, zA)

Textured

Return Type	**Name**	Arguments
Shape	**cone**	(x, y, z, r, h, xA, yA, zA, imageURL)

Cylinder

Draws a cylinder at (x,y,z) with radius r and height h. Can specify Euler rotation angles (xA, yA, zA). Can be solid, wireframe, or textured.

All Arguments

Type	**Name**	Description
double	**x**	x – coordinate of the center
double	**y**	y – coordinate of the center
double	**z**	z – coordinate of the center
double	**r**	radius
double	h	height
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation
String	**imageURL**	filename or URL of wraparound texture

Return Value

Returns the drawn Shape object, which can be further manipulated.

Solid

Return Type	**Name**	Arguments
Shape	**cylinder**	(x, y, z, r, h)
Shape	**cylinder**	(x, y, z, r, h, xA, yA, zA)

Wireframe

Return Type	**Name**	Arguments
Shape	**wireCylinder**	(x, y, z, r, h)
Shape	**wireCylinder**	(x, y, z, r, h, xA, yA, zA)

Textured

Return Type	**Name**	Arguments
Shape	**cylinder**	(x, y, z, r, h, xA, yA, zA, imageURL)

Ellipsoid

Draws an ellipsoid at (x, y, z) with dimensions (w, h, d). Can specify Euler rotation angles (xA, yA, zA). Can be solid, wireframe, or textured.

All Arguments

Type	**Name**	Description
double	**x**	x – co

double	**y**	y – coordinate of the center
double	**z**	z – coordinate of the center
double	**w**	width
double	h	height
double	d	depth
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation
String	**imageURL**	filename or URL of wraparound texture

Return Value

Returns the drawn Shape object, which can be further manipulated.

Solid

Return Type	**Name**	Arguments
Shape	**ellipsoid**	(x, y, z, w, h, d)
Shape	**ellipsoid**	(x, y, z, w, h, d, xA, yA, zA)

Wireframe

Return Type	**Name**	Arguments
Shape	**wireEllipsoid**	(x, y, z, w, h, d)
Shape	**wireEllipsoid**	(x, y, z, w, h, d, xA, yA, zA)

Textured

Return Type	**Name**	Arguments
Shape	**ellipsoid**	(x, y, z, w, h, d, xA, yA, zA, imageURL)

Advanced 3D Shapes

These more complicated 3D objects are useful in many cases. Every drawing function returns a Shape object, which can be moved, rotated, and scaled dynamically using the methods of the Shape class.

3D Text

Draws 3D text of the given string at (x, y, z) with the pen font. Can specify Euler rotation angles (xA, yA, zA). Uses the pen font, color, and transparency.

Arguments

Type	**Name**	Description
double	**x**	x – coordinate of the center
double	**y**	y – coordinate of the center
double	**z**	z – coordinate of the center
String	**text**	the text to be shown
double	**xA**	x component of XYZ Euler angle rotation
double	**yA**	y component of XYZ Euler angle rotation
double	**zA**	z component of XYZ Euler angle rotation

Return Value

Returns the drawn Shape object, which can be further manipulated.

Methods

Return Type	**Name**	Arguments
Shape	**text3D**	(x, y, z, text)
Shape	**text3D**	(x, y, z, text, xA, yA, zA)

Tubes

Draws a solid 3D cylindrical tube or a set of tubes with the given start and end coordinates and radius. Drawing a set of tubes is not much more efficient than drawing individual tubes, but will compact code. Can specify a set of colors for the set of tubes.

All Arguments

Type	**Name**	Description
double	**x1**	x – coordinate of the start point
double	**y1**	y – coordinate of the start point
double	**z1**	z – coordinate of the start point
double	**x2**	x – coordinate of the end point
double	**y2**	y – coordinate of the end point
double	**z2**	z – coordinate of the end point
double[]	**x**	set of x – coordinates of the tube vertices
double[]	**y**	set of y – coordinates of the tube vertices
double[]	**z**	set of z – coordinates of the tube vertices
double	**r**	radius of the tube
Color[]	**colors**	set of colors that correspond to the set of tubes

Return Value

Returns the drawn Shape object, which can be further manipulated.

Single Tube

Return Type	**Name**	Arguments
Shape	tube	(x1, y1, z1, x2, y2, z2, r)

Set of Tubes

Return Type	**Name**	Arguments
Shape	**tubes**	(x, y, z, r)
Shape	**tubes**	(x, y, z, r, colors)

External Model

You can easily import .OBJ (preferred) or .PLY models into your scene as Shape objects. It takes some time to create a mesh for larger models with millions of triangles. Colors and textures of an OBJ file can be imported by including the .mtl material template library. Can also import the model as a Shape with a single color (the pen color), and change the color using the Shape class. Imported OBJ models can be automatically scaled to have an average radius of 1, or keep their original size (default).

Note: The Java loader used to import OBJ files is based on old OBJ standards and often doesn't work properly for textures and newer commands in the OBJ file.

Arguments

Type	**Name**	Description
String	**filename**	filename of the model to import
boolean	resize	resizes the model if true

Return Value

Returns the drawn Shape object, which can be further manipulated.

Methods

Return Type	**Name**	Arguments
Shape	**model**	(filename)
Shape	**model**	(filename, resize)
Shape	**coloredModel**	(filename)
Shape	**coloredModel**	(filename, resize)

Display and Animation

The basic tools needed to display and animate a 3D scene are these functions for clearing or showing objects in the drawing window. If you are drawing a single scene without moving objects, just call *finished()* once after you are done drawing. If you are animating, call *show()* to display each frame with the desired framerate and *clear()* to clear the drawing window when desired. You can also clear just the 3D scene or just the 2D overlay, to enable independence between the two.

Showing

Show functions render everything to the drawing window, then pause for the given number of milliseconds. If no argument is provided, renders to the screen but does not pause. Methods are also provided to show just the 3D scene or to show just the 2D overlay. *finished()* is equivalent to *show(infinity)*.

Arguments

Type	**Name**	Description
int	**time**	pauses for this many milliseconds after displaying

Methods

Return Type	**Name**	Arguments
void	**finished**	()
void	**show**	(time)
void	**show**	()
void	**show3D**	(time)
void	**show3D**	()
void	**showOverlay**	(time)
void	**showOverlay**	()

Clearing

Clears the drawing window upon the next call of *show*. Methods are also provided to clear just the 3D scene or to clear just the 2D overlay.

Arguments

Type	Name	Description
Color	**color**	if provided, sets as the background color after clearing

Methods

Return Type	Name	Arguments
void	**clear**	(color)
void	**clear**	()
void	**clear3D**	()
void	**clearOverlay**	()

Static Camera Placement

The simplest way to programmatically place the camera where you want is to use these static camera methods. For more advanced camera control, learn how to use and manipulate the Camera object.

Camera Position and Orientation

Sets the camera position and orientation. Orientation is represented by XYZ Euler rotation angles. These numbers correspond to those shown in the info display on the upper-left corner of the drawing window. Can specify scalar or vector input. Can also get the current position and orientation.

All Arguments

Type	Name	Description
double	**x**	x – coordinate of camera position
double	**y**	y – coordinate of camera position
double	**z**	z – coordinate of camera position
Vector3D	**position**	position vector, represents {x, y, z}
double	**xAngle**	x component of XYZ Euler angle rotation
double	**yAngle**	y component of XYZ Euler angle rotation
double	**zAngle**	z component of XYZ Euler angle rotation
Vector3D	**angles**	orientation vector, represents {xAngle, yAngle, zAngle}

Position

Return Type	Name	Arguments
void	**setCameraPosition**	(x, y, z)
void	**setCameraPosition**	(position)
Vector3D	**getCameraPosition**	()

Orientation

Return Type	Name	Arguments

void	**setCameraOrientation** (xAngle, yAngle, zAngle)
void	**setCameraOrientation** (angles)
void	**getCameraOrientation** ()

Both

Return Type	Name	Arguments
void	**setCamera**	(x, y, z, xAngle, yAngle, zAngle)
void	**setCamera**	(position, angles)

Camera Direction

Instead of specifying orientation by Euler angles, it is easier in many cases to specify the vector direction in which the camera should point. For example, (0,1,0) would cause the camera to rotate such that it looks along the positive y-axis. Input is normalized. Can specify scalar or vector input. Can also get the current camera direction.

All Arguments

Type	Name	Description
double	**x**	x – component of camera direction vector
double	**y**	y – component of camera direction vector
double	**z**	z – component of camera direction vector
Vector3D	**direction**	camera direction vector, same as {x, y, z}

Position

Return Type	Name	Arguments
void	**setCameraDirection**	(x, y, z)
void	**setCameraDirection**	(direction)
Vector3D	**getCameraDirection**	()

Interactive Navigation

In StdDraw3D, you can move and navigate through the 3D scene interactively using mouse and keyboard controls. The five camera navigation modes and their controls are described below. If you are creating your own set of controls, use FIXED_MODE to disable interactivity. The camera mode can also be changed through the GUI menu, and with the keyboard shortcuts listed in that menu.

Mode Descriptions

StdDraw3D.ORBIT_MODE

Rotates and zooms about a central orbit center, which is the origin by default but can be changed.

Primary	Secondary	Effect
Left Mouse Drag		Orbit
Right Mouse Drag		Pan
Mouse Wheel	Alt Drag	Zoom

StdDraw3D.FPS_MODE

First-person-shooter style controls with arrow keys and WASD to move and mouse to look. The

positive y-axis is always upward.

Primary	Secondary	Effect
Up Arrow	W	Forward
Left Arrow	A	Left
Down Arrow	S	Backward
Right Arrow	D	Right
Page Up	Q	Up
Page Down	E	Down
Mouse Drag		Look

StdDraw3D.AIRPLANE_MODE

Similar to FPS_MODE, but the y-axis is not always upward, and can be rotated with Q and E.

Primary	Secondary	Effect
Up Arrow	W	Forward
Left Arrow	A	Left
Down Arrow	S	Backward
Right Arrow	D	Right
Page Up	Q	Rotate CCW
Page Down	E	Rotate CW
Mouse Drag		Look

StdDraw3D.LOOK_MODE

No interactive movement is allowed, but uses the mouse drag to look around.

Primary	Secondary	Effect
Mouse Drag		Look

StdDraw3D.FIXED_MODE

No interactive camera control is allowed. Use this if you want to only programmatically move your camera, or if you are defining your own set of navigation controls.

Setting the Camera Mode

Sets the camera navigation mode. If no argument is specified, reverts to the default of ORBIT_MODE. Can also get the current mode.

Arguments

Type	**Name**	Description
int	**mode**	one of the defined camera modes

Methods

Return Type	**Name**	Arguments
void	**setCameraMode**	(mode)
void	**setCameraMode**	()
int	**getCameraMode**	()

Orbit Center

ORBIT_MODE is the default mode, in which dragging with the left mouse button orbits about a

single point. This is the origin by default, but can be changed. Can specify scalar or vector input. Can also get the current orbit center.

All Arguments

Type	Name	Description
double	**x**	x – component of orbit center
double	**y**	y – component of orbit center
double	**z**	z – component of orbit center
Vector3D	**v**	orbit center, equivalent to {x, y, z}

Methods

Return Type	Name	Arguments
void	**setOrbitCenter**	(x, y, z)
void	**setOrbitCenter**	(v)
Vector3D	**getOrbitCenter**	()

Points, Lines, and Surfaces

Besides solid three-dimensional objects, you can also draw lower-dimensional shapes like points, lines, and polygons. Colors and thicknesses are still governed by pen settings. Methods for drawing sets of objects rather than individual ones are much more efficient. Every drawing function returns a Shape object, which can be moved, rotated, and scaled dynamically using the methods of the Shape class.

Points

Draws a single point or a set of points with the given coordinates. Drawing a set of points is much more efficient than drawing individual points, and you can still specify individual colors for each point. For example, the first point is at (x[0], y[0], z[0]) with color colors[0].

All Arguments

Type	Name	Description
double	**x**	x – coordinate
double	**y**	y – coordinate
double	**z**	z – coordinate
double[]	**x**	set of x – coordinates
double[]	y	set of y – coordinates
double[]	z	set of z – coordinates
Color[]	**colors**	set of colors that correspond to the set of points

Return Value

Returns the drawn Shape object, which can be further manipulated.

Single Point

Return Type	Name	Arguments
Shape	**point**	(x, y, z)

Set of Points

Return Type	**Name**	Arguments
Shape	**points**	(x, y, z)
Shape	**points**	(x, y, z, colors)

Lines

Draws a single line or a set of line with the given start and end coordinates. Drawing a set of lines is much more efficient than drawing individual lines, and you can still specify individual colors for each line. For example, the first line is from (x[0], y[0], z[0]) to (x[1], y[1], z[1]), and the second line is from (x[1], y[1], z[1]) to (x[2], y[2], z[2]). Vertex colors can be specified by the given array, and line colors are blends of its two vertex colors.

All Arguments

Type	**Name**	Description
double	**x1**	x – coordinate of the start point
double	**y1**	y – coordinate of the start point
double	**z1**	z – coordinate of the start point
double	**x2**	x – coordinate of the end point
double	**y2**	y – coordinate of the end point
double	**z2**	z – coordinate of the end point
double[]	**x**	set of x – coordinates of the line vertices
double[]	**y**	set of y – coordinates of the line vertices
double[]	**z**	set of z – coordinates of the line vertices
Color[]	**colors**	set of colors that correspond to the set of lines

Return Value

Returns the drawn Shape object, which can be further manipulated.

Single Line

Return Type	**Name**	Arguments
Shape	**line**	(x1, y1, z1, x2, y2, z2)

Set of Lines

Return Type	**Name**	Arguments
Shape	**lines**	(x, y, z)
Shape	**lines**	(x, y, z, colors)

Polygons

Draws a 2D polygon in 3D space with the given vertices. Vertices should be planar. Can specify filled or wireframe.

Arguments

Type	**Name**	Description
double[]	**x**	x – coordinates of each vertex
double[]	**y**	y – coordinates of each vertex
double[]	**z**	z – coordinates of each vertex

Return Value

Returns the drawn Shape object, which can be further manipulated.

Filled Polygon

Return Type	Name	Arguments
Shape	**polygon**	(x, y, z)

Wireframe Polygon

Return Type	Name	Arguments
Shape	**wirePolygon**	(x, y, z)

Triangles

Draws a set of triangles which are defined by the given coordinates. Coordinates are provided in an N by 9 array, where N is the number of triangles to draw and points[i] = {x1, y1, z1, x2, y2, z2, x3, y3, z3}. Can be used to create custom 3D meshes. Can specify filled or wireframe.

Arguments

Type	Name	Description
double[][]	**points**	set of nine vertex coordinates for N triangles
Color[]	**colors**	set of colors that correspond to the set of triangles

Return Value

Returns the drawn Shape object, which can be further manipulated.

Filled Triangles

Return Type	Name	Arguments
Shape	**triangles**	(points)
Shape	**triangles**	(points, colors)

Wireframe Triangles

Return Type	Name	Arguments
Shape	**wireTriangles**	(points)
Shape	**wireTriangles**	(points, colors)

Saving and Loading

You can save your current drawing window as an image file or a 3D scene. The 3D scene can be loaded later without having the original code that created it. You can also use the GUI toolbar menu of the drawing window or the keyboard shortcuts listed there.

Image File

Saves the current drawing window to a file. The specified file must have a .png or .jpg extension!

Arguments

Type	Name	Description

String	**filename**	name of the file to save to

Methods

Return Type	**Name**	Arguments
void	**save**	(filename)

3D Scene

There is experimental support for saving and loading a 3D scene as a file. When you save, the current 3D scene in the drawing window is written to a file which can be loaded back in. Some computers have been found to not load properly. The recommended extension for identification purposes is *.3d*.

Arguments

Type	**Name**	Description
String	**filename**	name of the file to save or load

Methods

Return Type	**Name**	Arguments
void	**saveScene3D**	(filename)
void	**loadScene3D**	(filename)

Lights

Light in the 3D scene is what controls the way they appear, and it is fully controllable. Light in StdDraw3D behaves similarly to actual light. At the initialization of each 3D scene, *setDefaultLight()* is called, which places two white directional lights in the scene - this is why a sphere has two bright sides and a dark band in the middle. You can only see object colors that can be reflected by the light color. Thus, white light makes any color visible, but if you only have red light in the scene, you will only see the red component of object colors.

Light Utilities

The default lighting consists of two white directional lights. If you want to set up custom lighting, then clear the existing lights with *clearLight()* and add your own.

Methods

Return Type	**Name**	Arguments
void	**setDefaultLight**	()
void	**clearLight**	()

Ambient Light

Ambient light is light that is everywhere, regardless of position or orientation. If there is ambient light, every object is lit up equally by it. Usually you want to keep ambient light very low, so pick a dark color or a very transparent one.

Arguments

Type	Name	Description
Color	**col**	color of the light

Return Value

Returns the rendered Light object, which can be further manipulated.

Methods

Return Type	Name	Arguments
Light	**ambientLight**	(col)

Point Light

Point lights emanate from a single point (x, y, z). They light up closer objects more than faraway objects. Can specify scalar or vector input. Can specify a power multiplier for the light, or go with a default of 1.0

All Arguments

Type	Name	Description
double	**x**	x – coordinate of the light source
double	**y**	y – coordinate of the light source
double	**z**	z – coordinate of the light source
Vector3D	**origin**	origin of the light source, equal to {x, y, z}
Color	**col**	color of the light
double	**power**	power multiplier of the light

Return Value

Returns the rendered Light object, which can be further manipulated.

Methods

Return Type	Name	Arguments
Light	**pointLight**	(x, y, z, col, power)
Light	**pointLight**	(x, y, z, col)
Light	**pointLight**	(origin, col, power)
Light	**pointLight**	(origin, col)

Directional Light

Directional lights shine in a single direction (x,y,z), but appear to come from far away. They will always light up exactly half of a sphere. Can specify scalar or vector input.

All Arguments

Type	Name	Description
double	**x**	x – component of the light direction
double	**y**	y – component of the light direction
double	**z**	z – component of the light direction
Vector3D	**dir**	direction of the light, equal to {x, y, z}
Color	**col**	color of the light

Return Value

Returns the rendered Light object, which can be further manipulated.

Methods

Return Type	Name	Arguments
Light	**directionalLight**	(x, y, z, col)
Light	**directionalLight**	(dir, col)

Fog

Fog makes objects that are far harder to see. You can only see fog in front of an object, you can't see fog obscuring empty space. You can control the strength and color of the fog.

All Arguments

Type	Name	Description
Color	**col**	color of the fog
double	**frontDistance**	distance from the camera where fog becomes noticable
double	**backDistance**	distance from the camera where fog is 100 % opaque

Methods

Return Type	Name	Arguments
void	**addFog**	(col, frontDistance, backDistance)
void	**clearFog**	()

Sounds

Create simple and efficient sound effects for interactive games or simulations, as well as background music. Sounds play in separate threads, and do not need time for preloading.

Sound Utilities

By default, StdDraw3D is silent. To stop a playing sound, clear the sound stack.

Methods

Return Type	Name	Arguments
void	**clearSound**	()

Ambient Sound

Ambient sound is heard everywhere in the 3D scene, regardless of position or orientation. Sound starts playing as soon as the function is called, and runs in a separate thread. Can specify to loop the sound infinitely until cleared.

All Arguments

Type	Name	Description
String	**filename**	audio file of sound to play
boolean	**loop**	option to loop the sound until cleared

Methods

Return Type	Name	Arguments
void	**playAmbientSound**	(filename)
void	**playAmbientSound**	(filename, loop)

Point Sound

Point sound emanates from a single origin, and is louder if you are near it and pointing toward it.

Unfortunately, there are currently unexpected problems that don't allow point sounds to work, so this feature is not supported. Any help would be appreciated.

Mouse and Keyboard

These functions give the state of the mouse and keyboard in the drawing window. They will not work when focus is outside the drawing window. They can be used to create new navigation controls or to provide interactivity to simulations, games, and models.

Mouse Buttons

Is the given mouse button (1 = left, 2 = right, 3 = middle) currently pressed down? Also included is a function that returns true if any mouse button is pressed.

Return Value

Returns the state of the mouse button.

Methods

Return Type	Name	Arguments
boolean	**mousePressed**	()
boolean	**mouse1Pressed**	()
boolean	**mouse2Pressed**	()
boolean	**mouse3Pressed**	()

Mouse Position

Return the current x and y position of the mouse, in terms of the coordinates defined by the scale (equal to the overlay drawing coordinates).

Return Value

Returns the x or y coordinate of the current mouse position.

Methods

Return Type	Name	Arguments
double	**mouseX**	()
double	**mouseY**	()

Key Pressed

Is the given key currently pressed down? The keys correnspond to physical keys, not characters. For letters, use the uppercase character to refer to the key. For arrow keys and modifiers such as shift and ctrl, refer to the KeyEvent constants, such as KeyEvent.VK_SHIFT.

Arguments

Type	**Name**	Description
int	**key**	the key of interest

Return Value

True if the given physical key is pressed.

Methods

Return Type	**Name**	Arguments
boolean	**isKeyPressed**	(key)

Key Stack

When a character is typed, it is added to the top of a character stack and kept track of. These methods let you find out when new keys have been typed and get those keys in the correct order. These typed values refer to characters, not to physical keys. For example, '6' will be different from '^'.

Return Value

hasNextKeyTyped() returns true if the user has typed a new character, and *nextKeyTyped()* returns the typed character.

Methods

Return Type	**Name**	Arguments
boolean	**hasNextKeyTyped**	()
char	**nextKeyTyped**	()

2D Overlay

You can draw 2D overlays on top of the 3D drawing window, to act as a heads up display or present informational text. All drawing functions of the Standard Draw library are available in StdDraw3D under the prefix of "*StdDraw3D.overlay"*. For example, use *StdDraw3D.overlayCircle(0, 0, 0.5)* instead of *StdDraw.circle(0,0,0.5)*. Overlay coordinates correspond to those set by *StdDraw3D.setScale()* but do not depend on the camera position or rotation.

Circle

Draws a circle of radius r, centered on (x, y). Can be an outline or filled.

Arguments

Type	**Name**	Description

double	**x**	x – coordinates of center
double	**y**	y – coordinates of center
double	**r**	radius

Methods

Return Type	**Name**	Arguments
void	**overlayCircle**	(x, y, r)
void	**overlayFilledCircle**	(x, y, r)

Square

Draws a square of side length 2r, centered on (x, y). Can be an outline or filled.

Arguments

Type	**Name**	Description
double	**x**	x – coordinates of center
double	**y**	y – coordinates of center
double	**r**	radius

Methods

Return Type	**Name**	Arguments
void	**overlaySquare**	(x, y, r)
void	**overlayFilledSquare**	(x, y, r)

Rectangle

Draws a rectangle of given half-width and half-height centered on (x, y). Can be an outline or filled.

Arguments

Type	**Name**	Description
double	**x**	x – coordinates of center
double	**y**	y – coordinates of center
double	**halfWidth**	half of the rectangle width
double	**halfHeight**	half of the rectangle height

Methods

Return Type	**Name**	Arguments
void	**overlayRectangle**	(x, y, halfWidth, halfHeight)
void	**overlayFilledRectangle**	(x, y, halfWidth, halfHeight)

Ellipse

Draws a filled ellipse with given semimajor and semiminor axes, centered on (x, y). Can be an outline or filled.

Arguments

Type	**Name**	Description
double	**x**	x – coordinates of center
double	**y**	y – coordinates of center
double	**semiMajorAxis**	ellipse semimajor axis
double	**semiMinorAxis**	ellipse semiminor axis

Methods

Return Type	**Name**	Arguments
void	**overlayEllipse**	(x, y, semiMajorAxis, semiMinorAxis)
void	**overlayFilledEllipse**	(x, y, semiMajorAxis, semiMinorAxis)

Pixel

Draws a single pixel at (x, y).

Arguments

Type	**Name**	Description
double	**x**	x – coordinate
double	**y**	y – coordinate

Methods

Return Type	**Name**	Arguments
void	**overlayPixel**	(x, y)

Point

Draws a small point at (x, y).

Arguments

Type	**Name**	Description
double	**x**	x – coordinate
double	**y**	y – coordinate

Methods

Return Type	**Name**	Arguments
void	**overlayPoint**	(x, y)

Line

Draws a line from (x0, y0) to (x1, y1).

Arguments

Type	**Name**	Description
double	**x0**	x – coordinate of the start point
double	**y0**	y – coordinate of the start point
double	**x1**	x – coordinate of the end point
double	**y1**	y – coordinate of the end point

Methods

Return Type	**Name**	Arguments
void	**overlayLine**	(x0, y0, x1, y1)

Arc

Draws an arc of radius r, centered on (x, y), from angle1 to angle2 (in degrees).

Arguments

Type	Name	Description
double	**x**	x – coordinates of center
double	**y**	y – coordinates of center
double	**r**	radius
double	**angle1**	starting arc angle
double	**angle2**	ending arc angle

Methods

Return Type	Name	Arguments
void	**overlayArc**	(x, y, r, angle1, angle2)

Polygon

Draws a polygon with the given vertex coordinates. Can be an outline or filled.

Arguments

Type	Name	Description
double[]	**x**	x – coordinates of vertices
double[]	**y**	y – coordinates of vertices

Methods

Return Type	Name	Arguments
void	**overlayPolygon**	(x, y)
void	**overlayFilledPolygon**	(x, y)

Text

Writes the given text string in the pen font and size, aligned about the point (x, y). Can be rotated. Useful for titles and HUD-style text in the drawing window.

All Arguments

Type	Name	Description
double	**x**	x – coordinate
double	**y**	y – coordinate
String	**text**	text to draw
double	**degrees**	angle of rotation

Methods

Return Type	Name	Arguments
void	**overlayText**	(x, y, text)
void	**overlayText**	(x, y, text, degrees)
void	**overlayTextLeft**	(x, y, text)
void	**overlayTextRight**	(x, y, text)

Picture

Draws a picture (gif, jpg, or png) centered on (x, y). Can rotated a given number of degrees, and rescaled to w-by-h.

All Arguments

Type	Name	Description
double	**x**	x – coordinate

double	**y**	y – coordinate
String	**s**	filename of image
double	**w**	width
double	**h**	height
double	**degrees**	angle of rotation

Methods

Return Type	Name	Arguments
void	**overlayPicture**	(x, y, s)
void	**overlayPicture**	(x, y, s, degrees)
void	**overlayPicture**	(x, y, s, w, h)
void	**overlayPicture**	(x, y, s, w, h, degrees)

Random Generators

For convenience, these methods which return commonly needed randomly generated parameters are included in the StdDraw3D library.

Color

Generates a random opaque color. Can choose either a fully random color or a random color on the rainbow.

Return Value

Returns a randomly generated color.

Methods

Return Type	Name	Arguments
Color	**randomColor**	()
Color	**randomRainbowColor**	()

Vector Direction

Returns a unit Vector3D in a random direction.

Return Value

Returns a random unit Vector3D.

Methods

Return Type	Name	Arguments
Vector3D	**randomDirection**	()

SHAPE CLASS

Everything three-dimensional you draw in your scene from spheres to points to 3D text is actually a Shape object. When you call a drawing function, it always returns a Shape object. If you keep a reference to this object, you can manipulate the already drawn Shape instead of clearing and redrawing it, which is a much more powerful and more efficient method of animation.

The Shape Class is referenced as *StdDraw3D.Shape*. There are no direct constructors or public fields - every Shape is created through a static drawing function. Here is an example:

```
StdDraw3D.Shape ball = StdDraw3D.sphere(0,0,0,1);
ball.move(0.2,0,0); // Moves the sphere
```

Static Shape Manipulation

These static functions are included in this section because they deal directly with Shape objects. They are powerful tools for creating efficient and advanced 3D scenes with StdDraw3D.

Combine

Combines any number of given Shape objects into one Shape object and returns it. Can input individual Shape objects separated by commas or an array of Shape objects. Useful for creating kinematic chains of relative motion

Arguments

Type	**Name**	Description
Shape([])	**shapes**	individual Shapes or an array of Shapes to copy

Return Value

Returns the combined Shape object.

Methods

Return Type	**Name**	Arguments
Shape	**combine**	(...shapes)

Copy

Returns an identical copy of a given Shape. The copy can be controlled independently. Much more efficient than redrawing a specific shape or model.

Arguments

Type	Name	Description
Shape	**shape**	3 D Shape to copy

Return Value

Returns a copy of the given Shape.

Methods

Return Type	Name	Arguments
Shape	**copy**	(shape)

Position

There are several ways to move a Shape object in 3D space. You can move it based on absolute coordinates or relative coordinates, or just set its position. For all methods, you can either specify scalar components of movement or a vector.

Move Absolute

Moves this Shape in absolute coordinates by the given vector.

All Arguments

Type	Name	Description
double	**x**	x – component of move vector
double	**y**	y – component of move vector
double	**z**	z – component of move vector
Vector3D	**move**	vector to move by, equal to {x, y, z}

Methods

Return Type	Name	Arguments
void	**move**	(x, y, z)
void	**move**	(move)

Move Relative

Moves this Shape in relative coordinates by the given vector, where (x, y, z) components correspond to (right, up, forward).

All Arguments

Type	Name	Description
double	**right**	x – component of move vector
double	**up**	y – component of move vector
double	**forward**	z – component of move vector
Vector3D	**move**	vector to move by, equal to {right, up, forward}

Methods

Return Type	Name	Arguments
void	**moveRelative**	(right, up, forward)
void	**moveRelative**	(move)

Set Position

Sets or gets the absolute position of this Shape.

All Arguments

Type	Name	Description
double	**x**	x – component of position vector
double	**y**	y – component of position vector
double	**z**	z – component of position vector
Vector3D	**pos**	new position vector, equal to {x, y, z}

Methods

Return Type	Name	Arguments
void	**setPosition**	(x, y, z)
void	**setPosition**	(pos)
Vector3D	**getPosition**	()

Orientation

There are many approaches to rotation in 3D space, and it is good to have options when orienting Shape objects. For example, you may want to rotate based on a relative or absolute frame, and you may prefer to use Euler angles, quaternions, axial rotations, or direction vectors. Each of these methods is useful in a different context, especially when making games. For most methods, you can either specify scalar components or a vector.

Rotate Absolute

Rotates this Shape by the given Euler angles (in degrees) in the absolute frame.

All Arguments

Type	Name	Description
double	**xAngle**	Euler rotation angle about x – axis
double	**yAngle**	Euler rotation angle about y – axis
double	**zAngle**	Euler rotation angle about z – axis
Vector3D	**angles**	Euler rotation angles

Methods

Return Type	Name	Arguments
void	**rotate**	(xAngle, yAngle, zAngle)
void	**rotate**	(angles)

Rotate Relative

Rotates this Shape by the given Euler angles (in degrees) in the relative frame.

All Arguments

Type	Name	Description
double	**pitch**	pitch component of Euler rotation
double	**yaw**	yaw component of Euler rotation

double	**roll**	roll component of Euler rotation
Vector3D	**angles**	vector to rotate by, equal to {x, y, z}

Methods

Return Type	Name	Arguments
void	**rotateRelative**	(pitch, yaw, roll)
void	**rotateRelative**	(angles)

Set Orientation

Sets or gets the absolute orientation of this Shape by the given Euler angles (in degrees).

All Arguments

Type	Name	Description
double	**xAngle**	Euler rotation angle about x – axis
double	**yAngle**	Euler rotation angle about y – axis
double	**zAngle**	Euler rotation angle about z – axis
Vector3D	**angles**	Euler rotation angles

Methods

Return Type	Name	Arguments
void	**setOrientation**	(xAngle, yAngle, zAngle)
void	**setOrientation**	(angles)
Vector3D	**getOrientation**	()

Rotate about Axis

Rotates this Shape about a given rotation axis by a certain number of degrees.

Arguments

Type	Name	Description
Vector3D	**axis**	vector that defines the axis of rotation
double	**angle**	angle of rotation (in degrees)

Methods

Return Type	Name	Arguments
void	**rotateAxis**	(axis, angle)

Look At

Sets this Shape to point toward a specific point in space. Can specify an up vector, or use the default of the positive y-axis. Useful for something like animating a pair of eyes that need to look at something else that is moving.

All Arguments

Type	Name	Description
Vector3D	**center**	point to look at
Vector3D	**up**	direction of up

Methods

Return Type	Name	Arguments

```
void        lookAt (center)
void        lookAt (center, up)
```

Set Direction

Sets or gets the vector direction that this Shape is currently pointing in. Can specify an up vector, or use the default of the positive y-axis. Useful for something like animating a cannon, which needs to point in a specific direction.

All Arguments

Type	Name	Description
Vector3D	**direction**	direction vector
Vector3D	**up**	direction of up

Methods

Return Type	Name	Arguments
void	**setDirection**	(direction)
void	**setDirection**	(direction, up)
Vector3D	**getDirection**	()

Size

Scales the size of this Shape relative to its current size. For example, *a.scale(2)* doubles the size of the Shape named *a*.

Arguments

Type	Name	Description
double	**scale**	scale multiplier

Methods

Return Type	Name	Arguments
void	**scale**	(scale)

Color

Sets the color of everything contained inside this Shape to the given color. The color change is recursive, and Shapes with multiple colors will become monotone. Can also specify transparency.

All Arguments

Type	Name	Description
Color	**c**	new color
int	**alpha**	transparency (0 – 255)

Methods

Return Type	Name	Arguments
void	**setColor**	(c)
void	**setColor**	(c, alpha)

Matching

Matches the position and orientation of this Shape with that of a given Shape or that of the Camera object.

All Arguments

Type	**Name**	Description
Shape	**s**	object of reference
Camera	**c**	object of reference

Methods

Return Type	**Name**	Arguments
void	**match**	(s)
void	**match**	(c)

Hiding

Hiding a Shape disappears it from the drawing window. While hidden, you can still manipulate the Shape as always, but you cannot see it. You can bring back the Shape by unhiding it. The methods *hide()* and *unhide()* are similar to *show()* and *clear()* for a single Shape object. Hiding is also useful for preloading large OBJ models so there is no load time when using them later.

Methods

Return Type	**Name**	Arguments
void	**hide**	()
void	**unhide**	()

CAMERA CLASS

The camera can be controlled with the static functions already listed in this reference. However, much more advanced control of the camera can be obtained by manipulating the Camera object. The Camera is basically equivalent to a Shape with a couple of extra functions, so it can be moved and rotated just like a Shape. This functionality works well with point-of-view simulations and games.

The Camera Class is referenced as *StdDraw3D.Camera*. There are no direct constructors or public fields - the only way to get the Camera is through the static function *camera()*. Here is an example:

```
StdDraw3D.Camera cam = StdDraw3D.camera();
cam.rotateRelative(0,10,0); // Rotates the view upward
```

Getting the Camera Object

This static function returns the Camera object, which represents the viewpoint of the drawing window. It can be manipulated in real time just like any other object in the 3D scene.

Return Value

Returns the Camera object.

Method

Return Type	**Name**	Arguments
Camera	**camera**	()

Methods from Shape Class

The Camera class has kinematic methods equivalent to those of the Shape class. A few methods are not available to the Camera class, because the Camera cannot be scaled, colored, or hidden. The Camera class has the following methods of the Shape class:

- move()
- moveRelative()
- setPosition()
- getPosition()
- rotate()
- rotateRelative()
- setOrientation()
- getOrientation()
- rotateAxis()

- lookAt()
- setDirection()
- getDirection()
- match()

Additional Methods

These methods are specifically for the Camera, and are useful in some scenarios.

Pair

Pairs the camera's position and orientation to those of a Shape, over time and motion. This is similar to calling *match()* for every frame of an animation, but may be less jumpy in some cases. Call *unpair()* to release the pairing.

Arguments

Type	**Name**	Description
Shape	**s**	object to pair with

Methods

Return Type	**Name**	Arguments
void	**pair**	(s)
void	**unpair**	()

Rotate FPS

Rotates the Camera by the given Euler angles (in degrees), but ensures that the positive y-axis is always in the upward direction and that the Camera does not point closer than 5 degrees from vertical. This is similar to the view control of most first person shooters.

All Arguments

Type	**Name**	Description
double	**xAngle**	rotation angle about x – axis
double	**yAngle**	rotation angle about y – axis
double	**zAngle**	rotation angle about z – axis
Vector3D	**angles**	rotation angles

Methods

Return Type	**Name**	Arguments
void	**rotateFPS**	(xAngle, yAngle, zAngle)
void	**rotateFPS**	(angles)

LIGHT CLASS

When you create a light in StdDraw3D, it returns a Light object. Light objects can be manipulated just like Shapes, and are useful if you want moving lights or lights that change color and brightness.

The Light Class is referenced as *StdDraw3D.Light*. There are no direct constructors or public fields - every Light is created through a static light function. Here is an example:

```
StdDraw3D.Light bulb = StdDraw3D.pointLight(2,1,3,StdDraw3D.RED);
bulb.setColor(StdDraw3D.BLUE); // Turns the red light blue
```

Methods from Shape Class

The Light class has kinematic methods equivalent to those of the Shape class. A light cannot be scaled, so *scale()* is not valid for Lights. The Light class has the following methods of the Shape class:

- `move()`
- `moveRelative()`
- `setPosition()`
- `getPosition()`
- `rotate()`
- `rotateRelative()`
- `setOrientation()`
- `getOrientation()`
- `rotateAxis()`
- `lookAt()`
- `setDirection()`
- `getDirection()`
- `setColor()`
- `match()`
- `hide()`
- `unhide()`

Note, the most important functions from these for a Light are *setColor()* and *hide()*.

Power of a Point Light

Point lights affect near objects more than faraway objects. The *setPower()* method allows control of the power of a point Light. Ambient and directional lights do not depend on distance, and their brightness is controlled by the light color only.

Arguments

Type	**Name**	Description
double	**power**	power of the point light

Method

Return Type	**Name**	Arguments
void	**scalePower**	(power)

VECTOR3D CLASS

Vector3D is an immutable three-dimensional vector class with useful vector operations like projection and reflection. It is used throughout StdDraw3D to deal with 3D points, vectors, and tuples. Technically, points are not be vectors and should have their own class, but they are represented with vectors in StdDraw3D for ease of use.

The Vector3D class is referenced as *StdDraw3D.Vector3D*. Here is an example of usage:

```
StdDraw3D.Vector3D p = new StdDraw3D.Vector3D (0,0.5,-2);
StdDraw3D.setCameraPosition (p);
```

Fields

Vector3D is composed of three double-precision components: x, y, and z. The values are immutable but directly accessible by calling the name of the vector followed by the field. For example:

```
StdDraw3D.Vector3D v = new StdDraw3D.Vector3D (1,2,3);
double xComp = v.x; // xComp now equals 1
```

Vector3D Fields

Type	**Name**	Description
double	**x**	x – component of this vector
double	**y**	y – component of this vector
double	**z**	z – component of this vector

Constructors

Creates a Vector3D object from three components or an array of components. If no arguments are given, initializes to the zero vector.

All Arguments

Type	**Name**	Description
double	**x**	x – component of the vector
double	**y**	y – component of the vector
double	**z**	z – component of the vector
double[]	**c**	array of three vector components, equal to {x, y, z}

Return Value

Newly created Vector3D.

Constructors

Return Type	Name	Arguments
Vector3D	**Vector3D**	(x, y, z)
Vector3D	**Vector3D**	(c)
Vector3D	**Vector3D**	()

Methods

Addition

Returns the sum of this vector and that vector. That vector can be either a Vector3D or three components.

All Arguments

Type	Name	Description
double	**x**	x-component of that vector
double	**y**	y-component of that vector
double	**z**	z-component of that vector
Vector3D	**that**	that vector, same as to {x, y, z}

Methods

Return Type	Name	Arguments
Vector3D	**plus**	(that)
Vector3D	**plus**	(x, y, z)

Subtraction

Returns the difference of this vector and that vector. That vector can be either a Vector3D or three components.

All Arguments

Type	Name	Description
double	**x**	x-component of that vector
double	**y**	y-component of that vector
double	**z**	z-component of that vector
Vector3D	**that**	that vector, same as to {x, y, z}

Methods

Return Type	Name	Arguments
Vector3D	**minus**	(that)
Vector3D	**minus**	(x, y, z)

Multiplication

Returns the product of this vector either uniformly with a single scalar or component-wise with three scalars.

All Arguments

Type	Name	Description
double	**a**	scalar to multiply x-component by

double **b** scalar to multiply y – component by
double **c** scalar to multiply z – component by
double **k** scalar to multiply each component by (ie. a = b = c)

Methods

Return Type	**Name**	Arguments
Vector3D	**times**	(k)
Vector3D	**times**	(a, b, c)

Magnitude

Returns the magnitude of this vector.

Methods

Return Type	**Name**	Arguments
double	**mag**	()

Direction

Returns the unit Vector3D in the direction of this vector.

Methods

Return Type	**Name**	Arguments
Vector3D	**direction**	()

Distance

Returns the Euclidian distance between this vector and that vector.

Arguments

Type	**Name**	Description
Vector3D	**that**	that vector

Methods

Return Type	**Name**	Arguments
double	**distanceTo**	(that)

Dot Product

Returns the dot product of this vector and that vector.

Arguments

Type	**Name**	Description
Vector3D	**that**	that vector

Methods

Return Type	**Name**	Arguments
double	**dot**	(that)

Cross Product

Returns the cross product of this vector and that vector.

Arguments

Type	**Name**	Description
Vector3D	**that**	that vector

Methods

Return Type	**Name**	Arguments
Vector3D	**cross**	(that)

Angle

Returns the smallest angle between this vector and that vector, in degrees.

Arguments

Type	**Name**	Description
Vector3D	**that**	that vector

Methods

Return Type	**Name**	Arguments
double	**angle**	(that)

Projection

Returns the projection of this vector onto the line defined by the given vector.

Arguments

Type	**Name**	Description
Vector3D	**line**	vector that defines the line of projection

Methods

Return Type	**Name**	Arguments
Vector3D	**proj**	(line)

Reflection

Returns the reflection of this vector across the line defined by the given vector.

Arguments

Type	**Name**	Description
Vector3D	**line**	vector that defines the line of reflection

Methods

Return Type	**Name**	Arguments
Vector3D	**reflect**	(line)

To String

Returns a formatted string representation of this vector "(this.x, this.y, this.z)".

Methods

Return Type	**Name**	Arguments
String	**toString**	()

Draw

Draws the tip of this vector from the origin as a sphere of radius 0.01.

Methods

Return Type	**Name**	Arguments
void	**draw**	()

This manual and StdDraw3D was written by Hayk Martirosyan in 2011. If you need help, found a bug, or have a question, feel free to email *hmartiro@princeton.edu*. If you have in interest in working on the project, let me know.