PyDy Visualization Distribution Documentation

Release 0.1.0

Tarun Gaba

February 22, 2014

This is the central page for all PyDyViz's Documentation. If you are already familiar with PyDy mechanics package(sympy.physics.mechanics), you can start with the Tutorial.

Contents:

CHAPTER 1

Introduction

PyDyViz (short for PythonDynamics Visualizations) is a plugin to facilitate browser based animations for PythonDynamics framework.

The plugin is used to generate animations for the multibody physical systems. The systems are defined in PyDy Mechanics, and solved numerically by PyDy Code Generator.

Frontend is based on WebGraphics Library (Three.js) and code generation is taken care of by PyDy.

The package also provides a Python wrapper for some basic functionality for Three.js i.e Geometries, Lights, Cameras etc.

CHAPTER 2

Installation

There are different methods you can use to get PyDyViz up and running for your system.

Dependencies:

1. Python
2. SymPy
3. NumPy

SciPy is used by the Code Generator module for numerical integration of Equations of Motions.So not exactly a dependency for PyDyViz, but is required anyways for Code Generator to work.

2.1 Source

PyDyViz can be installed from source via archive. Download the latest release archive from here.Extract the archive and cd into the directory. Issue the following command from command line:

```
$ python setup.py install
```

It should get the PyDyViz up and running.

2.2 Git

If you are a developer, or looking for the latest features without waiting for another release, you can install the development version from the git repository.

Issue the following command from terminal:

```
$ git clone git://github.com/PythonDynamics/pydy-viz.git
```

and then change directory to the cloned directory and run setup.py install:

```
$ cd pydy-viz
$ python setup.py install
```

and you will have the latest development version on board.

2.3 Python Package Index

You can also install PyDyViz from Python Package Index using pip or easy_install. Its as easy as opening a terminal and typing in:

```
$ pip install pydy_viz
```

or for easy_install:

```
$ easy_install pydy_viz
```

2.4 Run PyDyViz

You can check the installation by running the following command from Python Interpreter:

```
>>> import pydy_viz
```

If it does not throws any error/traceback, it means that PyDyViz is installed.

You can check the version of the software by issuing following command from interpreter:

```
>>> import pydy_viz
>>> print pydy_viz.__version__
```

2.5 Questions

If you have any question about installation, or any general question, feel free to visit the IRC channel at irc.freenode.net, channel #pydy. In addition, our mailing list is an excellent source of community support.

If you think there's a bug or you would like to request a feature, please open an issue.

CHAPTER 3

API

All the module specific docs have some test cases, which will prove helpful in understanding the usage of the particular module.

3.1 Python Modules Reference

3.1.1 Shapes

Shape

class pydy_viz.shapes.**Shape**(*name='unnamed'*, *color='grey'*)

A Shape. It is a superclass for more general shapes like Mesh, Cylinder, Sphere etc.

Default Shape can be used for Particle visualizations, as in sympy.physics.mechanics.particle

The Shape classes are used for creating visualization objects, used for studying multibody dynamics and in animations.

Values need to be supplied on initialization, but can be changed later.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for this shape

Examples

```
>>> from pydy_viz.shapes import Shape
>>>
>>> s = Shape()
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>#These can be changed later too ..
```

```
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> a = Shape('my-shape2', 'red')
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

color
color property of the shape, used for visualizations

color_in_rgb()
Returns the rgb value of the defined shape color.

generate_dict()
Generates data dict along with the Shape info to be used by VisualizationFrame class.

name
Name property of the shape, defines a name to a shape.

Cube

class pydy_viz.shapes.**Cube**(*name='unnamed'*, *color='grey'*, *length=10*)
A Cube. This class generates a Cube, with given length of side, and color.Default color is grey.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Cube

length: int or float. :

Length of side of Cube

Examples

```
>>> from pydy_viz.shapes import Cube
>>>
>>> s = Cube(10)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>s.length
10
>>>#These can be changed later too ..
```

```
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.length = 12
>>> s.length
12
>>> a = Cube('my-shape2', 'red', length=10)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.length
10
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cube, to be used by VisualizationFrame class.

Cylinder

class pydy_viz.shapes.**Cylinder**(*name='unnamed'*, *color='grey'*, *length=10*, *radius=5*)
A Cylinder. This class generates a Cylinder with given length, radius, and color. Default color is grey.

Parameters **name** : str

Name assigned to Cylinder

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Cylinder

length: int or float, length of the Cylinder :

radius: int or float, radius of the Cylinder :

Examples

```
>>> from pydy_viz.shapes import Cylinder
>>>
>>> s = Cylinder(length=10, radius=5)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.length
10
>>> s.radius
5
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
```

```
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.length = 12
>>> s.length
12
>>> s.radius = 6
>>> s.radius
6
>>> a = Cylinder('my-shape2', 'red', length=10, radius=5)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.length
10
>>> a.radius
5
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cylinder, to be used by VisualizationFrame class.

Cone

class pydy_viz.shapes.**Cone**(*name='unnamed'*, *color='grey'*, *length=10*, *radius=5*)
A Cone. This class generates a Cone with given length, base radius, and color. Default color is grey.

Parameters **name** : str

Name assigned to Cone

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Cone

length: int or float, length of the Cone :

radius: int or float, base radius of the Cone :

Examples

```
>>> from pydy_viz.shapes import Cone
>>>
>>> s = Cone(length=10, radius=5)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.length
10
>>> s.radius
5
```

```
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.length = 12
>>> s.length
12
>>> s.radius = 6
>>> s.radius
6
>>> a = Cone('my-shape2', 'red', length=10, radius=5)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.length
10
>>> a.radius
5
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cone, to be used by VisualizationFrame class.

Sphere

class pydy_viz.shapes.**Sphere**(*name='unnamed'*, *color='grey'*, *radius=10*)
A Sphere. This class generates a Sphere, with given length of side, and color. Default color is grey.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Sphere

radius: int or float. :

Radius of Sphere

Examples

```
>>> from pydy_viz.shapes import Sphere
>>>
>>> s = Sphere(10)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>s.radius
```

```
10
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.radius = 12
>>> s.radius
12
>>> a = Sphere('my-shape2', 'red', radius=10)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cube, to be used by VisualizationFrame class.

Circle

class pydy_viz.shapes.**Circle**(*name='unnamed'*, *color='grey'*, *radius=10*)
A Circle. This class generates a Circle, with given radius, and color. Default color is grey.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Circle

radius: int or float. :

Radius of Circle

Examples

```
>>> from pydy_viz.shapes import Circle
>>>
>>> s = Circle(10)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>s.radius
10
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
```

```
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.radius = 12
>>> s.radius
12
>>> a = Circle('my-shape2', 'red', radius=10)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Circle, to be used by VisualizationFrame class.

Plane

class pydy_viz.shapes.**Plane**(*name='unnamed'*, *color='grey'*, *length=10*, *width=5*)
A Plane. This class generates a Plane with given length, width, and color. Default color is grey.

Parameters **name** : str

Name assigned to Plane

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Plane

length: int or float, length of the Plane :

width: int or float, radius of the Plane :

Examples

```
>>> from pydy_viz.shapes import Plane
>>>
>>> s = Plane(length=10, width=5)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.length
10
>>> s.width
5
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
```

```
>>> s.color
'blue'
>>> s.length = 12
>>> s.length
12
>>> s.width = 6
>>> s.width
6
>>> a = Plane('my-shape2', 'red', length=10, width=5)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.length
10
>>> a.width
5
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Plane, to be used by VisualizationFrame class.

Tetrahedron

class pydy_viz.shapes.**Tetrahedron**(*name='unnamed'*, *color='grey'*, *radius=10*)
A Tetrahedron. This class generates a Tetrahedron. The argument given is the radius of the circumscribing sphere of the tetrahedron, and color. Default color is grey.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Tetrahedron

radius: int or float. :

Radius of circum-scribing sphere of Tetrahedron

Examples

```
>>> from pydy_viz.shapes import Tetrahedron
>>>
>>> s = Tetrahedron(10)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>s.radius
10
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
```

```
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.radius = 12
>>> s.radius
12
>>> a = Tetrahedron('my-shape2', 'red', radius=10)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cube, to be used by VisualizationFrame class.

Octahedron

class pydy_viz.shapes.**Tetrahedron**(*name='unnamed'*, *color='grey'*, *radius=10*)
A Tetrahedron. This class generates a Tetrahedron. The argument given is the radius of the circumscribing sphere of the tetrahedron, and color. Default color is grey.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Tetrahedron

radius: int or float. :

Radius of circum-scribing sphere of Tetrahedron

Examples

```
>>> from pydy_viz.shapes import Tetrahedron
>>>
>>> s = Tetrahedron(10)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>s.radius
10
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
```

```
>>> s.color
'blue'
>>> s.radius = 12
>>> s.radius
12
>>> a = Tetrahedron('my-shape2', 'red', radius=10)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cube, to be used by VisualizationFrame class.

Icosahedron

class pydy_viz.shapes.**Icosahedron**(*name='unnamed'*, *color='grey'*, *radius=10*)
A Icosahedron. This class generates a Icosahedron. The argument given is the radius of the circumscribing sphere of the icosahedron, and color. Default color is grey.

Parameters **name** : str

Name assigned to shape

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Icosahedron

radius: int or float. :

Radius of the circum-scribing sphere for Icosahedron

Examples

```
>>> from pydy_viz.shapes import Icosahedron
>>>
>>> s = Icosahedron(10)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>>s.radius
10
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
```

```
>>> s.radius = 12
>>> s.radius
12
>>> a = Icosahedron('my-shape2', 'red', radius=10)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Cube, to be used by VisualizationFrame class.

Torus

class pydy_viz.shapes.**Torus**(*name='unnamed'*, *color='grey'*, *radius=10*, *tube_radius=5*)
A Torus. This class generates a Torus with given radius, tube-radius, and color. Default color is grey.

Parameters **name** : str

Name assigned to Torus

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Torus

radius: int or float, radius of the Torus Shape :

tube_radius: int or float, radius of the torus tube :

Examples

```
>>> from pydy_viz.shapes import Torus
>>>
>>> s = Torus(radius=10, tube_radius=5)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.radius
10
>>> s.tube_radius
5
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.radius = 12
>>> s.radius
```

```
12
>>> s.tube_radius = 6
>>> s.tube_radius
6
>>> a = Torus('my-shape2', 'red', radius=10, tube_radius=5)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.tube_radius
5
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Torus, to be used by VisualizationFrame class.

TorusKnot

class pydy_viz.shapes.**TorusKnot**(*name='unnamed'*, *color='grey'*, *radius=10*, *tube_radius=5*)
A TorusKnot. This class generates a TorusKnot with given radius, tube-radius, and color. Default color is grey.

Parameters **name** : str

Name assigned to TorusKnot

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for TorusKnot

radius: int or float, radius of the TorusKnot Shape :

tube_radius: int or float, radius of the torus-knot tube :

Examples

```
>>> from pydy_viz.shapes import TorusKnot
>>>
>>> s = TorusKnot(radius=10, tube_radius=5)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.radius
10
>>> s.tube_radius
5
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
```

```
'blue'
>>> s.radius = 12
>>> s.radius
12
>>> s.tube_radius = 6
>>> s.tube_radius
6
>>> a = TorusKnot('my-shape2', 'red', radius=10, tube_radius=5)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
10
>>> a.tube_radius
5
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()

Generates data dict along with the Shape info for Torus, to be used by VisualizationFrame class.

Tube

class pydy_viz.shapes.**Tube**(*name='unnamed'*, *color='grey'*, *radius=10*, *points=None*)

A Tube. This class generates a Tube from given points, by drawing a curve passing through given points, with given radius and color. Default color is grey.

Parameters **name** : str

Name assigned to Tube

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Tube

radius: radius of Tube :

points: list of points which are used for making Tube :

Examples

```
>>> from pydy_viz.shapes import Tube
>>> point_list = [[1, 2, 1], [2, 1, 1], [2, 3, 4]]
>>> s = Tube(points=point_list)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.points
[[1, 2, 1], [2, 1, 1], [2, 3, 4]]
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
>>> s.name
```

```
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.radius = 14
>>> s.radius
14
>>> s.points = [[2, 1, 4], [1, 2, 4], [2, 3, 1], [1, 1, 3]]
>>> s.points
[[2, 1, 4], [1, 2, 4], [2, 3, 1], [1, 1, 3]]
>>> a = Tube('my-shape2', 'red', radius=12, points=point_list)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.radius
12
>>> a.points
[[1, 2, 1], [2, 1, 1], [2, 3, 4]]
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Tube, to be used by VisualizationFrame class.

Mesh

class pydy_viz.shapes.**Mesh**(*name='unnamed'*, *color='grey'*, *points=None*)
A Mesh. This class generates a general Mesh from given points, by drawing a curve passing through given points, and color. Default color is grey.

Parameters **name** : str

Name assigned to Mesh

color: str :

A color string from list of colors in pydy_viz.colors module This color is used in drawing visualizations for Mesh

points: list of points which are used for making mesh :

Examples

```
>>> from pydy_viz.shapes import Mesh
>>> point_list = [[1, 2, 1], [2, 1, 1], [2, 3, 4]]
>>> s = Mesh(points=point_list)
>>> s.name
'unnamed'
>>> s.color
'grey'
>>> s.color_in_rgb()
(0.5019607843137255, 0.5019607843137255, 0.5019607843137255)
>>> s.points
[[1, 2, 1], [2, 1, 1], [2, 3, 4]]
>>>#These can be changed later too ..
>>> s.name = 'my-shape1'
```

```
>>> s.name
'my-shape1'
>>> s.color = 'blue'
>>> s.color
'blue'
>>> s.points = [[2, 1, 4], [1, 2, 4], [2, 3, 1], [1, 1, 3]]
>>> s.points
[[2, 1, 4], [1, 2, 4], [2, 3, 1], [1, 1, 3]]
>>> a = Mesh('my-shape2', 'red', points=point_list)
>>> a.name
'my-shape2'
>>> a.color
'red'
>>> a.points
[[1, 2, 1], [2, 1, 1], [2, 3, 4]]
>>> a.color_in_rgb()
(1.0, 0.0, 0.0)
```

generate_dict()
Generates data dict along with the Shape info for Mesh, to be used by VisualizationFrame class.

3.1.2 VisualizationFrame

class pydy_viz.**VisualizationFrame**(**args*)
A VisualizationFrame represents an object that you want to visualize. It allows you to easily associate a reference frame and a point with a shape.

A VisualizationFrame can be attached to only one Shape Object. It can be nested, i.e we can add/remove multiple visualization frames to one visualization frame. On adding the parent frame to the Scene object, all the children of the parent visualization frame are also added, and hence can be visualized and animated.

A VisualizationFrame needs to have a ReferenceFrame, and a Point for it to form transformation matrices for visualization and animations.

The ReferenceFrame and Point are required to be provided during initialization. They can be supplied in the form of any one of these:

1)reference_frame, point argument. 2)a RigidBody argument 3)reference_frame, particle argument.

In addition to these arguments, A shape argument is also required.

evaluate_transformation_matrix(*dynamic_values*, *constant_values*)
Returns the numerical transformation matrices for each time step.

Parameters **dynamic_values** : array_like, shape(m,) or shape(n, m)

The m state values for each n time step.

constant_values : array_like, shape(p,)

The p constant parameter values of the system.

Returns **transform_matrix** : numpy.array, shape(n, 4, 4)

A 4 x 4 transformation matrix for each time step.

generate_numeric_transform_function(*dynamic_variables*, *constant_variables*)
Returns a function which returns a transformation matrix given the symbolic states and the symbolic system parameters.

Parameters **dynamic_variables** : list of all the dynamic symbols used in defining the

mechanics objects.

constant_variables : list of all symbols used in defining the

mechanics objects

Returns A Lambda function which returns a transformation matrix, :

given symbolic states, and symbolic system parameters :

generate_transformation_matrix(*reference_frame*, *point*)

Generates the symbolic Transformation matrix, with respect to the reference_frame, point in the argument.

Parameters reference_frame : ReferenceFrame

A reference_frame with respect to which transformation matrix :

is generated. :

point : Point

A point with respect to which transformation matrix :

is generated. :

Returns A SymPy 4by4 matrix, containing symbolic variables for :

transformation. :

generate_visualization_dict()

Returns a dictionary of all the info required for the visualization of this frame, alongwith child.

Before calling this method, all the transformation matrix generation methods should be called, or it will give an error.

Returns a dictionary containing following keys: :

name : name of the VisualizationFrame

children : simulation dictionary of child frames

shape : shape info of the attached shape,

like dimensions, color etc.It is generated from generator method :

of Shape class. :

simulation_matrix : a N*4*4 matrix, converted to list, for

passing to Javascript for animation purposes, where N is the :

number of timesteps for animations. :

name

Name of the VisualizationFrame.

origin

Origin of the VisualizationFrame, with respect to which all translational transformations take place.

reference_frame

reference_frame of the VisualizationFrame, with respect to which all rotational/orientational transformations take place.

shape

shape in the VisualizationFrame. A shape attached to the visualization frame. NOTE: Only one shape can be attached to a visualization frame.

3.1.3 Cameras

Perspective Camera

class pydy_viz.camera.**PerspectiveCamera**(**args*, ***kwargs*)

Creates a Perspective Camera for visualization. The camera is inherited from VisualizationFrame,

It can be attached to dynamics objects, hence we can get a moving camera. All the transformation matrix generation methods are applicable to a Perspective Camera. Like VisualizationFrame, It can also be initialized using: 1)Rigidbody 2)ReferenceFrame, Point 3)ReferenceFrame, Particle Either one of these must be supplied during initialization

Unlike VisualizationFrame, It doesnt require a Shape argument.

Parameters **name** : str

a name for the PerspectiveCamera(optional). Default is 'unnamed' :

fov : int or float

Field Of View, It determines the angle between the top and bottom :

of the viewable area(in degrees). Default is 45 (degrees) :

near : int or float

The distance of near plane of the PerspectiveCamera. :

All objects closer to this distance are not displayed. :

far : int or float

The distance of far plane of the PerspectiveCamera :

All objects farther than this distance are not displayed. :

far

attribute for Far Plane distance of a PerspectiveCamera Default is 1000

fov

attribute for Field Of view of a PerspectiveCamera Default is 45 degrees

generate_visualization_dict()

Returns a dictionary of all the info required for the visualization of this Camera

Before calling this method, all the transformation matrix generation methods should be called, or it will give an error.

Returns **a dictionary containing following keys:** :

name : name of the PerspectiveCamera

fov : Field Of View of the PerspectiveCamera

simulation_matrix : a N*4*4 matrix, converted to list, for

passing to Javascript for animation purposes, where N is the :

number of timesteps for animations. :

near

attribute for Near Plane distance of a PerspectiveCamera Default is 1

Orthographic Camera

class pydy_viz.camera.**OrthoGraphicCamera**(**args*, ***kwargs*)

Creates a OrthoGraphic Camera for visualization. The camera is inherited from VisualizationFrame,

It can be attached to dynamics objects, hence we can get a moving camera. All the transformation matrix generation methods are applicable to a Perspective Camera. Like VisualizationFrame, It can also be initialized using: 1)Rigidbody 2)ReferenceFrame, Point 3)ReferenceFrame, Particle Either one of these must be supplied during initialization

Unlike VisualizationFrame, It doesnt require a Shape argument.

Parameters **name** : str

a name for the PerspectiveCamera(optional). Default is 'unnamed' :

near : int or float

The distance of near plane of the PerspectiveCamera. :

All objects closer to this distance are not displayed. :

far : int or float

The distance of far plane of the PerspectiveCamera :

All objects farther than this distance are not displayed. :

far

attribute for Far Plane distance of an OrthoGraphicCamera Default is 1000

generate_visualization_dict()

Returns a dictionary of all the info required for the visualization of this Camera

Before calling this method, all the transformation matrix generation methods should be called, or it will give an error.

Returns **a dictionary containing following keys:** :

name : name of the OrthoGraphicCamera

simulation_matrix : a N*4*4 matrix, converted to list, for

passing to Javascript for animation purposes, where N is the :

number of timesteps for animations. :

near

attribute for Near Plane distance of an OrthoGraphicCamera Default is 1

3.1.4 Lights

PointLight

class pydy_viz.light.**PointLight**(**args*, ***kwargs*)

Creates a PointLight for the visualization The PointLight is inherited from VisualizationFrame,

It can also be attached to dynamics objects, hence we can get a moving Light. All the transformation matrix generation methods are applicable to a PointLight. Like VisualizationFrame, It can also be initialized using: 1)Rigidbody 2)ReferenceFrame, Point 3)ReferenceFrame, Particle Either one of these must be supplied during initialization

Unlike VisualizationFrame, It doesnt require a Shape argument.

color
Color of Light.

color_in_rgb()
Returns the rgb value of the defined light color.

generate_visualization_dict()
Returns a dictionary of all the info required for the visualization of this PointLight.

Before calling this method, all the transformation matrix generation methods should be called, or it will give an error.

Returns a dictionary containing following keys: :

name : name of the PointLight

color : color of the light used.

simulation_matrix : a N*4*4 matrix, converted to list, for

passing to Javascript for animation purposes, where N is the :

number of timesteps for animations. :

3.1.5 Scene

class pydy_viz.scene.**Scene**(*reference_frame*, *origin*, **visualization_frames*, ***kwargs*)
Scene class holds all the data required for the visualizations/ animation of a system.

It has methods for inputting the numerical data from the numerical integrations of Equations of Motions and convert them to JSON values, which can be then parsed by Javascripts(webgls).

A scene object takes a ReferenceFrame, and a Point as required arguments. The reference_frame and point act as the inertial frame and origin with respect to which all objects are oriented and rendered in the visualizations

A scene needs to be supplied with visualization_frames, Cameras, and Light objects, as optional arguments. A scene can also be supplied with the height and width of the browser window where visualization would be displayed. Default is 800 * 800.

display()
display method can be used in two ways. When called from IPython notebook, it shows the visualization in the form of output cell in the IPython notebook. If it is called from python interpreter or IPython interpreter(not notebook), It generates an html file, in the current directory, which can be opened in the webgl compliant browser for viewing the visualizations.

The simulation data is used from this scene, hence all simulation data generation methods should be called before calling this method

generate_visualization_dict(*dynamic_variables*, *constant_variables*, *dynamic_values*, *constant_values*)
generate_visualization_dict() method generates a dictionary of visualization data

Parameters dynamic_variables : Sympifyable list or tuple

This contains all the dynamic symbols or state variables which are required for solving the transformation matrices of all the frames of the scene.

constant_variables : Sympifyable list or tuple

This contains all the symbols for the parameters which are used for defining various objects in the system.

dynamic_values : list or tuple

initial states of the system. The list or tuple should be respective to the state_sym.

constant_values : list or tuple

values of the parameters. The list or tuple should be respective to the par_sym.

Returns The dictionary contains following keys: :

1) Width of the scene. :

2) Height of the scene. :

3) name of the scene. :

4) frames in the scene, which contains sub-dictionaries :

of all the visualization frames information.

generate_visualization_json(*dynamic_variables*, *constant_variables*, *dynamic_values*, *constant_values*, *save_to='data.json'*)

generate_visualization_json() method generates a json str, which is saved to file.

Parameters dynamic_variables : Sympifyable list or tuple

This contains all the dynamic symbols or state variables which are required for solving the transformation matrices of all the frames of the scene.

constant_variables : Sympifyable list or tuple

This contains all the symbols for the parameters which are used for defining various objects in the system.

dynamic_values : list or tuple

initial states of the system. The list or tuple should be respective to the state_sym.

constant_values : list or tuple

values of the parameters. The list or tuple should be respective to the par_sym.

save_to : str

path to the file where to write the generated data JSON. the path should be chosen such as to have the write permissions to the user.

Returns The dictionary contains following keys: :

1) Width of the scene. :

2) Height of the scene. :

3) name of the scene. :

4) frames in the scene, which contains sub-dictionaries :

of all the visualization frames information.

name

Returns Name of Scene.

origin

returns Origin of the Scene.

reference_frame

returns reference_frame of the Scene.

3.2 JavaScript functions Reference

3.2.1 Canvas

Canvas is the base class for handling all the animation and visualization generation.

3.2.2 canvas/initialize.js

Constructor:

This function acts as a class constructor for Canvas class It takes the JSON Object variable as the argument, which contains all the data in the JSON format. It binds onClick methods of certain Divs on the frontend with some Canvas.prototype functions.

Canvas.prototype.initialize

This prototype function initializes the starting canvas, on which all the visualizations are drawn.

It adds following to the canvas:

- A Primary Camera
- Primary Trackball Controls
- A Primary Light
- Axes
- Grid
- A Div for displaying total number of frames.
- A Div for displaying the current frame animation is running on.

3.2.3 canvas/addObjects.js

Canvas.prototype.addControls

This prototype function initializes the Primary Controls, which were defined in Canvas.prototype.initialize function.

It generates a controlsID, which contains the return value of requestAnimationFrame, and can be used to call cancelAnimationFrame, for stopping mouse controlled animation.

Canvas.prototype.resetControls

This prototype function simply calls the controls reset method for the Primary Controls(canvas.prototype.primaryControls).

Canvas.prototype.addCameras

This prototype function parses the JSON Object for cameras and adds them to the scene. All the cameras are stored in a Canvas.cameras object, which is an instance of THREE.Object3D();

Canvas.prototype.addLights

This prototype function parses the JSON Object for lights and adds them to the scene. All the lights are stored in a Canvas.lights object, which is an instance of THREE.Object3D();

Canvas.prototype.addFrames

This prototype function parses the JSON Object for frames and adds them to the scene. All the frames are stored in a Canvas.frames object, which is an instance of THREE.Object3D();

3.2.4 canvas/animate.js

Canvas.prototype.startAnimation

This prototype function kick starts the animation. It iterates over the frames and apply transformation matrices from Simulation Matrix of that frame, iteratively. By default animation is done for a single loop, which can be changed to looped by the check button from the UI.

Canvas.prototype.pauseAnimation

This prototype function pauses the animation, but retains the current animation frame.

Canvas.prototype.stopAnimation

This prototype function stops the animation, and resets current animation frame to 0.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*