STLIB Documentation

Release 1.0

damien.marchal@univ-lille1.fr

Aug 28, 2020

Contents

Python side of the framework Sofa.

Example:

```
TODO
```

splib.algorithms	Algorithms we often use.
splib.animation	Animation framework focusing in ease of use.
splib.debug	
splib.geometric	
splib.loaders	Utilitary function to ease the writing of scenes.
splib.numerics	
splib.objectmodel	
splib.scenegraph	Algorithms we often use.
splib.units	Toolbox to manipulats units (time, mechanics, material)

CHAPTER 1

splib.algorithms

Algorithms we often use.

Content:

CHAPTER 2

splib.animation

Animation framework focusing in ease of use.

2.1 Functions:

animate(onUpdate, params, duration[, mode, ...])	Construct and starts an animation
AnimationManager(node)	A Controller to manage all animations in the scene
AnimationManagerController	MagicMock is a subclass of Mock with default implementations of most of the magic methods.

2.1.1 splib.animation.animate

splib.animation.**animate**(*onUpdate*, *params*, *duration*, *mode='once'*, *onDone=None*)

Construct and starts an animation

Build a new animation from a callback function that computes the animation value, a set of parameters, the animation duration and the type of animation repetition pattern.

Animation can be added from any code location (createScene, PythonScriptController)

Parameters

- **duration** (*float*) – duration of the animation in seconds.
- **mode** (*str*) – once, loop, pingpong

Example:

```
def myAnimate(target, factor):
    print("I should do something on: "+target.name)


def createScene(rootNode)
```

(continues on next page)

(continued from previous page)

```
    AnimationManager(rootNode)
    animate(myAnimate, {"target" : rootNode }, 10)
```

2.1.2 splib.animation.AnimationManager

splib.animation.**AnimationManager**(*node*)

A Controller to manage all animations in the scene

Before using the animation framework an AnimationManager must be added to the scene. It has in charge, at each time step to update all the running animations.

Returns: AnimationManagerController

Example:

```
def createScene(rootNode)
    AnimationManager(rootNode)
```

2.1.3 splib.animation.AnimationManagerController

splib.animation.**AnimationManagerController**

2.2 Modules:

splib.animation.easing	Easing function to use in animation

2.2.1 splib.animation.easing

Easing function to use in animation

LinearRamp([beginValue, endValue, scale])	Linear interpolation between two values

splib.animation.easing.LinearRamp

splib.animation.easing.**LinearRamp**(*beginValue=0.0*, *endValue=1.0*, *scale=0.5*)

Linear interpolation between two values

Examples: LinearRamp(10, 20, 0.5)

CHAPTER 3

splib.loaders

Utilitary function to ease the writing of scenes.

loadPointListFromFile(s)	Load a set of 3D point from a json file
getLoadingLocation(filename[, source])	Compute a loading path for the provided filename relative to a given source location

3.1 splib.loaders.loadPointListFromFile

splib.loaders.**loadPointListFromFile**(*s*)
Load a set of 3D point from a json file

splib.loaders.**getLoadingLocation**(*filename*, *source=None*)
Compute a loading path for the provided filename relative to a given source location

Examples:

getLoadingLocation("myfile.json") # returns "myfile.json getLoadingLocation("myfile.json", "toto") #returns "/fullpath/to/toto/myfile.json" getLoadingLocation("myfile.json", __file__) #returns "/fullpath/to/toto/myfile.json"

The latter is really usefull to make get the path for a file relative to the 'current' python source.

CHAPTER 4

splib.objectmodel

CHAPTER 5

splib.scenegraph

Algorithms we often use.

Content:

find(node, path)	Query a node or an object by its path from the provided node.
get(node, path)	Query a node, an object or a data by its path from the provided node.

5.1 splib.scenegraph.find

splib.scenegraph.**find**(*node*, *path*)

Query a node or an object by its path from the provided node.

Example: find(node, "/root/rigidcube1/visual/OglModel")

5.2 splib.scenegraph.get

splib.scenegraph.**get**(*node*, *path*)

Query a node, an object or a data by its path from the provided node.

Example: find(node, "/root/rigidcube1/visual/OglModel.position")

CHAPTER 6

splib.units

Toolbox to manipulats units (time, mechanics, material)

Content:

time	Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)
material	Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)
units	Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

6.1 splib.units.time

Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

For each conversion a local unit can be given OR variables local_time/local_length/local_mass are used by default (these variables can be set by the user to define local units)

Note that Poisson ratio and strain have no units

6.2 splib.units.material

Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

For each conversion a local unit can be given OR variables local_time/local_length/local_mass are used by default (these variables can be set by the user to define local units)

Note that Poisson ratio and strain have no units

6.3 splib.units.units

Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

For each conversion a local unit can be given OR variables local_time/local_length/local_mass are used by default (these variables can be set by the user to define local units)

Note that Poisson ratio and strain have no units

- genindex
- modindex
- search

CHAPTER 7

Sofa Template Library

Utility functions and scene templates for the real-time simulation framework Sofa. The different templates are organized in types and abstract the complexity of object creation with Sofa.

The library is hosted on *github https://github.com/SofaDefrost/STLIB/* and it can be used with scenes written in python and PSL.

Example:

```
from stlib.scene import MainHeader
from stlib.solver import DefaultSolver
from stlib.physics.rigid import Cube, Sphere, Floor
from stlib.physics.deformable import ElasticMaterialObject

def createScene(rootNode):
    MainHeader(rootNode)
    DefaultSolver(rootNode)

    Sphere(rootNode, name="sphere", translation=[-5.0, 0.0, 0.0])
    Cube(rootNode, name="cube", translation=[5.0,0.0,0.0])

    ElasticMaterialObject(rootNode, name="dragon",
                          volumeMeshFileName="mesh/liver.msh",
                          surfaceMeshFileName="mesh/dragon.stl"
                          translation=[0.0,0.0,0.0])

    Floor(rootNode, name="plane", translation=[0.0, -1.0, 0.0])
```

Content:

stlib.physics	Templates for physical simulation.
stlib.visuals	Templates for rendering.
stlib.solver	Templates for most of the common time integration setups.

Continued on next page

Table 1 – continued from previous page

stlib.scene	Templates for most of the common scene setups.
stlib.tools	

7.1 stlib.physics

Templates for physical simulation.

Content:

stlib.physics.constraints	Templates for external constraints.
stlib.physics.collision	Templates to ease collision and contact handling.
stlib.physics.deformable	Templates for deformable objects.
stlib.physics.mixedmaterial	
stlib.physics.rigid	This package is focused on implementing a standard rigid object for sofa.

7.1.1 stlib.physics.constraints

Templates for external constraints.

Content

FixedBox([applyTo, atPositions, name, ...])	Constraint a set of degree of freedom to be at a fixed position.
PartiallyFixedBox([attachedTo, box, ...])	Constraint a set of degree of freedom to be at a fixed position.
SubTopology([attachedTo, containerLink, ...])	Args:

stlib.physics.constraints.FixedBox

stlib.physics.constraints.**FixedBox**(*applyTo=None, atPositions=[-1.0, -1.0, -1.0, 1.0, 1.0, 1.0], name='FixedBox', doVisualization=False, position=None, constraintStrength='1e12', doRecomputeDuringSimulation=False*)

Constraint a set of degree of freedom to be at a fixed position.

Args: applyTo (Sofa.Node): Node where the constraint will be applyied

atPosition (vec6f): Specify min/max points of the font.

name (str): Set the name of the FixedBox constraint.

doVisualization (bool): Control whether or not we display the boxes.

Structure:

```
Node : {
    name : "fixedbox",
```

(continues on next page)

(continued from previous page)

```
    BoxROI,
    RestShapeSpringsFroceField
}
```

stlib.physics.constraints.PartiallyFixedBox

stlib.physics.constraints.**PartiallyFixedBox**(*attachedTo=None, box=[-1.0, -1.0, -1.0, 1.0, 1.0, 1.0], fixedAxis=[1, 1, 1], name='PartiallyFixedBox', drawBoxes=False, fixAll=False, doUpdate=False*)

Constraint a set of degree of freedom to be at a fixed position.

Args: attachedTo (Sofa.Node): Node where the constraint will be applyied

box (vec6f): Specify min/max points of the font.

fixedAxis (vec3bool): Specify which axis should be fixed (x,y,z)

name (str): Set the name of the FixedBox constraint.

drawBoxes (bool): Control whether or not we display the boxes.

fixAll (bool): If true will apply the partial fixed to all the points.

doUpdate (bool): If true

Structure:

```
Node : {
    name : "fixedbox",
    BoxROI,
    PartialFixedConstraint
}
```

stlib.physics.constraints.SubTopology

stlib.physics.constraints.**SubTopology**(*attachedTo=None, containerLink=None, boxRoiLink=None, linkType='tetrahedron', name='modelSubTopo', poissonRatio=0.3, youngModulus=18000*)

Args:

attachedTo (Sofa.Node): Where the node is created.

containerLink (str): path to container with the dataField to link, ie: '@../container.position'

boxRoiLink (str): path to boxRoi with the dataField to link,

linkType (str): indicate of which type is the subTopo, ei: 'triangle' -> TriangleSetTopologyContainer & TriangleFEMForceField

name (str): name of the child node

youngModulus (float): The young modulus.

poissonRatio (float): The poisson parameter.

Structure:

```
Node : {
    name : "modelSubTopo",
    TetrahedronSetTopologyContainer,
    TetrahedronFEMForceField
}
```

7.1.2 stlib.physics.collision

Templates to ease collision and contact handling.

Content:

CollisionMesh([attachedTo, ...])	

stlib.physics.collision.CollisionMesh

stlib.physics.collision.**CollisionMesh**(*attachedTo=None, surfaceMeshFileName=None, name='collision', rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], collisionGroup=None, mappingType='BarycentricMapping'*)

7.1.3 stlib.physics.deformable

Templates for deformable objects.

Content:

ElasticMaterialObject	Creates an object composed of an elastic material.

stlib.physics.deformable.ElasticMaterialObject

class stlib.physics.deformable.**ElasticMaterialObject**(*attachedTo=None, volumeMeshFileName=None, name='ElasticMaterialObject', rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], scale=[1.0, 1.0, 1.0], surfaceMeshFileName=None, collisionMesh=None, withConstrain=True, surfaceColor=[1.0, 1.0, 1.0], poissonRatio=0.3, youngModulus=18000, totalMass=1.0, solver=None*)

Creates an object composed of an elastic material.

createPrefab(*volumeMeshFileName=None, name='ElasticMaterialObject', rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], scale=[1.0, 1.0, 1.0], surfaceMeshFileName=None, collisionMesh=None, withConstrain=True, surfaceColor=[1.0, 1.0, 1.0], poissonRatio=0.3, youngModulus=18000, totalMass=1.0, solver=None*)

addCollisionModel(*collisionMesh, rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], scale=[1.0, 1.0, 1.0]*)

addVisualModel(*filename, color, rotation, translation, scale=[1.0, 1.0, 1.0]*)

cls
alias of *ElasticMaterialObject*

definedloc = ('/home/docs/checkouts/readthedocs.org/user_builds/stlib/checkouts/stable

7.1.4 stlib.physics.rigid

This package is focused on implementing a standard rigid object for sofa. The object is made described by its surface mesh.

Content:

RigidObject(node[, name, ...])	Creates and adds rigid body from a surface mesh.
Cube(node, **kwargs)	Create a rigid cube of unit dimension
Sphere(node, **kwargs)	Create a rigid sphere of unit dimension
Floor(node, **kwargs)	Create a rigid floor of unit dimension

stlib.physics.rigid.RigidObject

stlib.physics.rigid.**RigidObject**(*node, name='RigidObject', surfaceMeshFileName=None, translation=[0.0, 0.0, 0.0], rotation=[0.0, 0.0, 0.0], uniformScale=1.0, totalMass=1.0, volume=1.0, inertiaMatrix=[1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0], color=[1.0, 1.0, 0.0], isAStaticObject=False*)

Creates and adds rigid body from a surface mesh.

Args: surfaceMeshFileName (str): The path or filename pointing to surface mesh file.

totalMass (float): The mass is distributed according to the geometry of the object.

color (vec3f): The default color used for the rendering of the object.

translation (vec3f): Apply a 3D translation to the object.

rotation (vec3f): Apply a 3D rotation to the object in Euler angles.

uniformScale (vec3f): Apply a uniform scaling to the object.

isAStaticObject (bool): The object does not move in the scene (e.g. floor, wall) but react to collision.

Structure:

```
Node : {
    name : "rigidobject"
    MechanicalObject,
    UniformMass,
    UncoupledConstraintCorrection,
    *EulerImplicit,
    *SparseLDLSolver,
```

(continues on next page)

(continued from previous page)

```
    Node : {
        name : "collision",
        Mesh,
        MechanicalObject,
        Triangle,
        Line,
        Point,
        RigidMapping
    }
    Node : {
       name : "visual"
       OglModel,
       RigidMapping
    }
}
```

stlib.physics.rigid.Cube

stlib.physics.rigid.**Cube**(*node*, ***kwargs*)
Create a rigid cube of unit dimension

stlib.physics.rigid.Sphere

stlib.physics.rigid.**Sphere**(*node*, ***kwargs*)
Create a rigid sphere of unit dimension

stlib.physics.rigid.Floor

stlib.physics.rigid.**Floor**(*node*, ***kwargs*)
Create a rigid floor of unit dimension

7.2 stlib.visuals

Templates for rendering.

7.3 stlib.solver

Templates for most of the common time integration setups.

Content:

DefaultSolver(node[, iterative])	Adds EulerImplicit, CGLinearSolver

7.3.1 stlib.solver.DefaultSolver

stlib.solver.**DefaultSolver**(*node*, *iterative=True*)

Adds EulerImplicit, CGLinearSolver

Components added: EulerImplicit CGLinearSolver

7.4 stlib.scene

Templates for most of the common scene setups.

Content:

Scene	Scene(SofaObject) Create a scene with default properties.
MainHeader(node[, gravity, dt, plugins, ...])	Args:
ContactHeader(applyTo, alarmDistance, ... [, ...])	Args:
Node(parentNode, name)	Create a new node in the graph and attach it to a parent node.
Wrapper(node, attachedFunction, datacache)	Args:

7.4.1 stlib.scene.Scene

class stlib.scene.**Scene**(*SofaObject*)

Create a scene with default properties.

Arg:

node (Sofa.Node) the node where the scene will be attached

gravity (vec3f) the gravity of the scene

dt (float) the dt time

plugins (list(str)) set of plugins that are used in this scene

repositoryPath (list(str)) set of path where to read the data from

doDebug (bool) activate debugging facility (to print text)

There is method to add default solver and default contact management on demand.

addSolver()

addContact(*alarmDistance*, *contactDistance*, *frictionCoef=0.0*)

cls

alias of *Scene*

definedloc = ('/home/docs/checkouts/readthedocs.org/user_builds/stlib/checkouts/stable

7.4.2 stlib.scene.Interaction

class stlib.scene.**Interaction**(*parent*, *targets*)

Store a list of mechanical object to interact with

Args:

targets ([objects]) the object to interact with and that don't have a solver

cls

alias of *Interaction*

definedloc = ('/home/docs/checkouts/readthedocs.org/user_builds/stlib/checkouts/stable

7.4.3 stlib.scene.MainHeader

stlib.scene.**MainHeader**(*node, gravity=[0.0, -9.8, 0.0], dt=0.01, plugins=[], repositoryPaths=[], doDebug=False*)

Args: gravity (vec3f): define the gravity vector.

dt (float): define the timestep.

plugins (list str): list of plugins to load

repositoryPaths (list str): list of path to the specific data repository

Structure:

```
rootNode : {
    gravity : gravity,
    dt : dt,
    VisualStyle,
    RepositoryPath,
    RequiredPlugin,
    OglSceneFrame,
    FreeMotionAnimationLoop,
    GenericConstraintSolver,
    DiscreteIntersection
}
```

7.4.4 stlib.scene.ContactHeader

stlib.scene.**ContactHeader**(*applyTo*, *alarmDistance*, *contactDistance*, *frictionCoef=0.0*)

Args: applyTo (Sofa.Node): the node to attach the object to

alarmDistance (float): define the distance at which the contact are integrated into the detection computation.

contactDistance (float): define the distance at which the contact response is integrated into the computation.

frictionCoef (float, default=0.0): optional value, set to non-zero to enable a global friction in your scene.

Structure:

```
rootNode : {
    CollisionPipeline,
    BruteForceDetection,
    RuleBasedContactManager,
    LocalMinDistance
}
```

7.4.5 stlib.scene.Node

stlib.scene.**Node**(*parentNode*, *name*)

Create a new node in the graph and attach it to a parent node.

7.4.6 stlib.scene.Wrapper

class stlib.scene.**Wrapper**(*node*, *attachedFunction*, *datacache*)

Args: node : the current node we are working on

attachedFunction [the function that will be called at each object creation] to do some stuff replace/insert/delete ...

This function will take as arguments the node, the type of the object to create, datacache & also the current arguments of the object .

This function as to return a tuple containing parameters of the object we want to create (ie: his type and a dictionary with all the other arguments) or None

datacache : the data we will use in our attachedFunction as parameters or else

createObject(*type*, ***kwargs*)

createChild(*name*)

__getattr__(*value*)

7.5 stlib.tools

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

C

D

E

F

G

I

L

M

N

P

R

S

W