ONERA

THE FRENCH AEROSPACE LAB

CPlot Documentation

Release 3.5

/ELSA/MU-10020/V3.5

Nov 21, 2022

CONTENTS

CHAPTER

ONE

PREAMBLE

CPlot is a simple plotter for arrays (as defined in Converter documentation) or for CGNS/python trees (pyTrees as defined in Converter/Internal documentation).

This module is part of Cassiopee, a free open-source pre- and post-processor for CFD simulations.

For use with the array interface, you have to import CPlot module:

```
import CPlot
```

For use with the pyTree interface:

```
import CPlot.PyTree as CPlot
```

CHAPTER

TWO

LIST OF FUNCTIONS

– Actions

CPlot.display(arrays[, dim, mode, ...])	Display arrays.
CPlot.render()	Force render.
CPlot.delete(zlist)	Delete zones from plotter.
CPlot.add(arrays, no, array[, zoneName, ...])	Add one zone to plotter.
CPlot.replace(arrays, no, array[, zoneName, ...])	Replace arrays[no] by array.
CPlot.finalizeExport([action])	Finalize export for continuous export.

– Set / Get functions

CPlot.getState(mode)	Return a state in plotter.
CPlot.getSelectedZone()	Return the selected zone in plotter.
CPlot.getSelectedZones()	Return the selected zones in plotter.
CPlot.getSelectedStatus(zone)	Return the selected status of a zone in plotter.
CPlot.getActiveZones()	Return the active (displayed) zones in plotter.
CPlot.getActiveStatus(zone)	Return the active status of a zone in plotter.
CPlot.getActivePoint()	Return the active (clicked) point in plotter.
CPlot.getActivePointIndex()	Return the active (clicked) point index.
CPlot.getMouseState()	Return mouse state (mouse position and button state).
CPlot.getKeyboard()	Return the pressed keys.
CPlot.resetKeyboard()	Reset the keyboard string.
CPlot.changeVariable()	Change displayed variable.
CPlot.changeStyle()	Change CPlot display style.
CPlot.changeBlanking()	Change the blanking procedure.

Continued on next page

Table 2 – continued from previous page

CPlot.setState([dim, mode, scalarField, ...])	Set CPlot state.
CPlot.setMode(mode)	Set CPlot display mode.
CPlot.setSelectedZones(zlist)	Set selected zones.
CPlot.unselectAllZones()	Unselect all zones.

– Camera setting and motion

CPlot.lookFor()	Look for selected zones.
CPlot.moveCamera(posCams[, posEyes, ...])	Move posCam and posEye following check points.
CPlot.travelLeft([xr, N])	Travel camera left.

– Set rendering informations in pyTree

CPlot.PyTree.addRender2Zone(t[, material, ...])	Add a renderInfo node to a zone node.
CPlot.PyTree.addRender2PyTree(t[, slot, ...])	Add a renderInfo node to a tree.
CPlot.PyTree.loadView(t[, slot])	Load a view stored in slot.

CHAPTER

THREE

CONTENTS

3.1 Keys in CPlot window

Keys must be pressed when CPlot window is active.

- **f**: fit view to data.
- **Ctrl+f**: switch between full screen and windowed mode.
- **Left/right Arrows** or **left mouse drag**: rotate model.
- **Up/down Arrows**: Zoom in/out.
- **Shift + Arrows** or **right mouse drag**: translate model.
- **Ctrl + right mouse drag**: tilt model.
- **Shift + left mouse click**: select zone.
- **Shift + Ctrl + left mouse click**: multiple select.
- **Ctrl + left mouse click**: Accurate select (click on nearest mesh node)
- **Shift + right mouse click**: deactivate (hide) zone.
- **Shift + double left mouse click**: center view on clicked point.
- **o** or **left mouse drag**: rotate model up.
- **p** or **left mouse drag**: rotate model down.
- **1** or **Shift+1**: display fields (switch variable - next and previous).
- **Space bar**: display mesh.
- **Shift+Space bar**: display solid.
- **m** or **M**: toggle between 2D and 3D mode.
- **z** or **Z**: select zones one by one.
- **a** or **A**: activate(show)/deactivate(hide) a selected zone.

- **l**: look for selected zone.
- **i** or **I** or **Ctrl+i** or **Ctrl+I**: change displayed i plane (structured zones).
- **j** or **J** or **Ctrl+j** or **Ctrl+J**: change displayed j plane.
- **k** or **K** or **Ctrl+k** or **Ctrl+K**: change displayed k plane.
- **q**: quit.

3.2 Actions

CPlot.**display**(*a*, ...)

Display entity. Display function has a lot of optional options that can be specified as arguments. In offscreen mode, you can render with openGL if you have a GPU or with osmesa if you only have a CPU.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **dim** (int) – dimension of data. 1: 1D, 2: 2D, 3: 3D (default: 3)
- **mode** (int or string) – display mode. 0 or 'Mesh': mesh, 1 or 'Solid': solid, 2 or 'Render': render, 3 or 'Scalar': scalar field, 4 or 'Vector': vector field (default: 0)
- **scalarField** (int or string) – scalar field number or scalar field name (ex:'Density')
- **vectorField1,2,3** (int or string) – vector field number or vector field name
- **displayInfo** (int) – 0 means no info display (default: 1)
- **displayIsoLegend** (int) – 0 means no iso legend display (default: 0)
- **meshStyle** (int) – 0: white solid and red wireframe, 1: colored wireframe, 2: colored solid and wireframe, 3: cyan solid and black wireframe (default: 2)
- **solidStyle** (int) – 0: blue, 1: colored by zone, 3: white, 4: colored by zone outlined (default: 1)
- **scalarStyle** (int) – 0: banded, 1: banded+mesh, 2: lines, 3: lines+mesh (default: 0)
- **vectorStyle** (int) – 0: RGB, 1: arrows, 2: lines (default: 0)

- **vectorDensity** (float) – the density of vectors (default: 0.)
- **vectorScale** (float in 0-100) – scale of vector in % (default: 100.)
- **vectorNormalize** (0 or 1) – if 1, displayed vectors are normalized (default: 0)
- **vectorShowSurface** (0 or 1) – if 1, display surface in vector mode (vectorStyle=1) (default: 1)
- **vectorShape** (0 (3D arrows), 1 (Flat arrows), 2 (Tetra arrows)) – type of arrows for vectors (vectorStyle=1) (default: 0)
- **vectorProjection** (0 or 1) – 1 of vectors are projected on surface (default: 0)
- **colormap** (int (upper number activates light)) – 0-1: Blue2Red, 2-3: BiColorRGB, 4-5: BiColorHSV, 6-7: TriColorRGB, 8-9: TriColorHSV, 10-11: MultiColorRGB, 12-13: MultiColorHSV, 14-15: Diverging, 16-17: Viridis, 18-19: Inferno, 20-21: Magma, 22-23: Plasma, 24-25: Jet, 26-27: Greys, 28-29: Nice Blue, 30-31: Greens (default: 0)
- **colormapC1** (string) – Hexa string for starting color of bi/tri colors colormaps (ex: #FFFFFF)
- **colormapC2** (string) – Hexa string for ending color of bi/tri colors colormaps (ex: #FFFFFF)
- **colormapC3** (string) – Hexa string for mid color of tri colors colormaps (ex: #FFFFFF)
- **niso** (int) – number of isos (default: 25)
- **isoEdges** (float) – width of iso edges for scalar display (default: -1)
- **isoScales** (list of [string, int, float, float] or [string, int, float, float, float, float] Additional colormap number can be added.) – list of min and max of a variable ([varName, niso, min, max] or [varName, niso, min, max, cutmin, cutmax])(default: [])
- **win** (tuple of 2 ints) – (sizeWinX, sizeWinY) window size (default: 700,700)
- **posCam** (tuple of 3 floats) – (x,y,z) camera position
- **posEye** (tuple of 3 floats) – (x,y,z) eye position
- **dirCam** (tuple of 3 floats) – (x,y,z) camera direction (default: 0,0,1)

- **viewAngle** (float) – camera angle in degrees (default: 50.)
- **bgColor** (int) – background color. 0-13 (default: 0)
- **shadow** (int) – display shadows. 0-1 (default: 0)
- **dof** (int) – depth of field smoothing. 0-1 (default: 0)
- **stereo** (int) – 1 or 2 means red/cyan anaglyph (default: 0)
- **stereoDist** (float) – distance between eyes for stereo
- **export** (string) – file name for export (.png)
- **exportResolution** (string) – resolution for export ("1920x1080")
- **zoneNames** (list of strings) – optional list of zone names (same size as arrays, struct zones, then unstruct zones)
- **renderTags** (list of strings) – optional list of render tags (same size as arrays, struct zones, then unstruct zones)
- **frameBuffer** (int) – the number of the frame buffer we are rendering to for offscreen rendering
- **offscreen** (int) – 0: rendering, 1: offscreen rendering (osmesa), 2: offscreen rendering (openGL), 3: partial composite offscreen rendering (openGL), 4: final composite offscreen rendering (openGL), 5: partial composite offscreen rendering (osmesa), 6: final composite offscreen rendering (osmesa), 7: parallel offscreen rendering (osmesa) (default: 0)

Example of use:

- Display (array):

```
# - display (array) -
import Generator as G
import CPlot
import Transform as T

a = G.cart((0,0,0),(1,1,1),(18,28,3))
CPlot.display(a, mode='mesh')

for i in range(360):
    a = T.rotate(a, (9, 14, 3.5), (0,0,1), 1.)
    CPlot.display(a)
```

- Display (pyTree):

```
# - display (pyTree) -
import Generator.PyTree as G
import CPlot.PyTree as CPlot
import Transform.PyTree as T

a = G.cart((0,0,0),(1,1,1),(18,28,3))

for i in range(360):
    a = T.rotate(a, (9, 14, 3.5), (0,0,1), 1.)
    CPlot.display(a)
```

- Display offscreen with openGL (pyTree):

```
# - display (pyTree) -
# display offscreen using GL
import CPlot.PyTree as CPlot
import Geom.PyTree as D

a = D.sphere((0,0,0), 1)
b = D.sphere((1,1,1), 1.2)

# One image, all scene
CPlot.display([a,b], mode='mesh',
              posCam=(0,6,0), posEye=(0,0,0), dirCam=(0,0,1),
              export="one.png", offscreen=2)
CPlot.finalizeExport()

# Compositing in one image
CPlot.display(a, mode='mesh',
              posCam=(0,6,0), posEye=(0,0,0), dirCam=(0,0,1),
              export="composite.png", offscreen=3)
CPlot.finalizeExport(3)
CPlot.display(b, mode='mesh',
              posCam=(0,6,0), posEye=(0,0,0), dirCam=(0,0,1),
              export="composite.png", offscreen=4)
CPlot.finalizeExport(4)

import os; os._exit(0)
```

- Display offscreen with osmesa (pyTree):

```
# - display (pyTree) -
# display offscreen using osmesa
import CPlot.PyTree as CPlot
```

(continues on next page)

(continued from previous page)

```
import Geom.PyTree as D

a = D.sphere((0,0,0), 1)
b = D.sphere((1,1,1), 1.2)

# One image, all scene
CPlot.display([a,b], mode='mesh',
              posCam=(0,6,0), posEye=(0,0,0), dirCam=(0,0,1),
              export="one.png", offscreen=1, exportResolution='3840x2160')

# Compositing in one image
CPlot.display(a, mode='mesh',
              posCam=(0,6,0), posEye=(0,0,0), dirCam=(0,0,1),
              export="composite.png", offscreen=5)
CPlot.display(b, mode='mesh',
              posCam=(0,6,0), posEye=(0,0,0), dirCam=(0,0,1),
              export="composite.png", offscreen=6)
```

CPlot.**render**()

Force rendering. Must be used after functions that don't render (ex: add, delete, ...).

Example of use:

- Render (array):

```
# - render (array) -
import Generator as G
import CPlot

a = G.cart((0,0,0), (1,1,1), (10,10,10))
CPlot.display(a)
CPlot.render()
```

- Render (pyTree):

```
# - render (pyTree) -
import Generator.PyTree as G
import CPlot.PyTree

a = G.cart((0,0,0), (1,1,1), (10,10,10))
CPlot.PyTree.display([a])
CPlot.PyTree.render()
```

CPlot.**delete**(*list*)

Delete zones from plotter. This function does not render. Argument is either a list of zone numbers (struct zones then unstructured zones order) or a list of zone names if zoneNames argument has been provided before to display function.

Parameters **list** (list of int or strings) – list of zone number or zone names

Example of use:

- Delete zones (array):

```
# - delete (array) -
import Generator as G
import CPlot
import time

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cart( (11,0,0), (1,1,1), (10,10,10) )

CPlot.display([a,b]); time.sleep(1)
CPlot.delete([0]); CPlot.render(); time.sleep(1)
CPlot.delete(['S-Zone 1']); CPlot.render(); time.sleep(1)
```

- Delete zones (pyTree):

```
# - delete (pyTree) -
import Generator.PyTree as G
import CPlot.PyTree as CPlot
import Converter.PyTree as C
import time

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cartTetra( (11,0,0), (1,1,1), (10,10,10) )
c = G.cart( (0,11,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base', a, b, c])

CPlot.display(t); time.sleep(1)
CPlot.delete(['Base/cartTetra']); CPlot.render(); time.sleep(1)
```

CPlot.**add**(*A*, ..., *a*)

Add/insert one array/zone in plotter. This function does not render.

For array interface:

```
CPlot.add(A, no, a)
```

no is the position of insertion of a in A. Replace also in A.

For the pyTree interface:

```
CPlot.add(A, nob, noz, a)
```

Insert or append a to the base nob, at position noz in the zone list. If noz=-1, append to end of list.

Parameters

- **A** (arrays, pyTree or list of zones) – input data
- **no** (int) – position of zone to add in A
- **nob** (int) – position of base of zone to add in A
- **noz** (int) – position of zone to add in A
- **a** (array or zone) – data to add

Example of use:

- Add a zone (array):

```
# - add (array) -
import Generator as G
import CPlot
import time

a = G.cart((0,0,0), (1,1,1), (30,30,30))
A = [a]
CPlot.display(A); time.sleep(1)

b = G.cart((30,0,0), (1,1,1), (30,30,30))
CPlot.add(A, 0, b); CPlot.render(); time.sleep(0.5)
```

- Add zone (pyTree):

```
# - add (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import CPlot.PyTree as CPlot
import time

a = G.cart((0,0,0), (1,1,1), (30,30,30))
```

(continues on next page)

(continued from previous page)

```
t = C.newPyTree(['Base', a])
CPlot.display(t); time.sleep(1)

b = G.cart((30,0,0), (1,1,1), (30,30,30))
CPlot.add(t, 1, -1, b); CPlot.render(); time.sleep(1)
```

CPlot.**replace**(*A*, ..., *a*)

For array interface:

```
CPlot.replace(A, no, a)
```

Performs A[no]=a, keeping plotter coherent.

For pyTree interface:

```
CPlot.replace(A, nob, noz, a)
```

Performs t[2][nob][2][noz]=a, keeping plotter coherent. This function does not render.

Parameters

- **A** (arrays, pyTree or zones) – input data
- **no** (int) – position of zone to add in A
- **nob** (int) – position of base of zone to add in A
- **noz** (int) – position of zone to add in A
- **a** (array or zone) – data to add

Example of use:

- Replace a zone (array):

```
# - replace (array) -
import Generator as G
import CPlot
import time

a = G.cart( (0,0,0), (1,1,1), (30,30,30) )
A = [a]
CPlot.display(A); time.sleep(1)

for i in range(10):
    b = G.cart( (i,0,0), (1,1,1), (30,30,30) )
    CPlot.replace(A, 0, b); CPlot.render(); time.sleep(0.1)
```

- Replace a zone (pyTree):

```
# - replace (pyTree) -
import Generator.PyTree as G
import CPlot.PyTree as CPlot
import Converter.PyTree as C
import time

a = G.cartTetra( (0,0,0), (1,1,1), (10,10,10) )
b = G.cartTetra( (11,0,0), (1,1,1), (10,10,10) )
c = G.cart( (0,11,0), (1,1,1), (10,10,10) )

t = C.newPyTree(['Base', a, b, c])
CPlot.display(t); time.sleep(2.)

d = G.cart( (11,11,0), (1,1,1), (10,10,10) )
CPlot.replace(t, 1, 0, d); CPlot.render(); time.sleep(1.)
```

CPlot.**finalizeExport**(*action=0*)

Finalize an export. Wait for the end of file writing (action=0) or in mpeg export, close the mpeg file (action=1). Must be called when doing offscreen rendering (offscreen=1 or 2 in display function).

Parameters **action** (int) – if 0, means wait until file is written, if 1, means close mpeg file

Example of use:

- Finalize an export (array):

```
# - display (array) -
# display offscreen using GL
import CPlot
import Transform as T
import Geom as D

a = D.sphere((0,0,0), 1, N=200)

# Multi images
CPlot.display(a, offscreen=2, bgColor=1, mode=0, meshStyle=2,
              solidStyle=1, posCam=(0,6,0), export='one.png')
CPlot.finalizeExport() # wait for end of file write
for i in range(5):
    a = T.rotate(a, (0,0,0), (0,0,1), 1.)
    CPlot.display(a, offscreen=2, bgColor=1, mode=0, meshStyle=2,
              solidStyle=1, posCam=(0,6,0), export='one%d.png'%i)
```

(continues on next page)

(continued from previous page)

```
    CPlot.finalizeExport() # wait for end of file write
import os; os._exit(0)

# Mpeg Movie
for i in range(50):
    a = T.rotate(a, (0,0,0), (0,0,1), 1.)
    CPlot.display(a, bgColor=1, mode=0,
                  solidStyle=1, posCam=(0,6,0), export='export.mpeg',
                  exportResolution='680x600', offscreen=2)
    CPlot.finalizeExport() # wait for end of file write
CPlot.finalizeExport(1) # close mpeg file
import os; os._exit(0)
```

3.3 Set / Get functions

CPlot.**getState**(*stateName*)

Return the specified state value as stored in plotter. Available stateName are the same as the display function arguments.

Parameters **stateName** (string) – name of state to be retrieved

Example of use:

- Get state from plotter (array):

```
# - getState (array) -
import Generator as G
import CPlot

a = G.cart((0,0,0), (1,1,1), (5,5,5))
CPlot.display(a)

print('dim=',CPlot.getState('dim'))
print('mode=',CPlot.getState('mode'))
print('displayInfo=',CPlot.getState('displayInfo'))
print('meshStyle=',CPlot.getState('meshStyle'))
print('solidStyle=',CPlot.getState('solidStyle'))
print('isoEdges=',CPlot.getState('isoEdges'))
print('win=',CPlot.getState('win'))
print('posCam=',CPlot.getState('posCam'))
print('posEye=',CPlot.getState('posEye'))
print('dirCam=',CPlot.getState('dirCam'))
```

- Get state from plotter (pyTree):

```
# - getState (pyTree) -
import Generator.PyTree as G
import CPlot.PyTree as CPlot

a = G.cart((0,0,0), (1,1,1), (5,5,5))
CPlot.display(a)

print('dim=',CPlot.getState('dim'))
print('mode=',CPlot.getState('mode'))
print('displayInfo=',CPlot.getState('displayInfo'))
print('meshStyle=',CPlot.getState('meshStyle'))
print('solidStyle=',CPlot.getState('solidStyle'))
print('isoEdges=',CPlot.getState('isoEdges'))
print('win=',CPlot.getState('win'))
print('posCam=',CPlot.getState('posCam'))
print('posEye=',CPlot.getState('posEye'))
print('dirCam=',CPlot.getState('dirCam'))
```

CPlot.**getSelectedZone**()

Return the currently selected zone. If none is selected, return -1. If multiple zones are selected, return the last selected zone.

Returns no of selected zone

Return type int

Example of use:

- Get selected zone (array):

```
# - getSelectedZone (array) -
import Generator as G
import CPlot
import time

a = G.cart((0,0,0), (1,1,1), (5,5,5))
CPlot.display(a)

nz = -1
while nz == -1:
    nz = CPlot.getSelectedZone(); time.sleep(0.1)
print('One zone has been selected: %d.'%nz)
```

CPlot.**getSelectedZones**()

Return the list of selected zones. If none is selected, return [].

Example of use:

- Get selected zones (array):

```
# - getSelectedZones (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

ret = []
while ret == []:
    ret = CPlot.getSelectedZones(); time.sleep(2.)
print('Zones have been selected: ', ret)
```

CPlot.**getSelectedStatus**(*nz*)

Return the selected status (1: selected, 0: not selected) of zone nz.

Parameters **nz** (int) – zone number

Returns status of zone (1: selected, 0: not selected)

Return type int

Example of use:

- Get selected status of zone (array):

```
# - getSelectedStatus (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(5.)
ret = CPlot.getSelectedStatus(0)
if ret == 1: print('Zone 0 is selected.')
else: print('Zone 0 is not selected.')
```

CPlot.**getActiveZones**()

Return the list of active (visible) zones.

Returns list of zone numbers

Return type list of ints

Example of use:

- Get active zones (array):

```
# - getActiveZones (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(5.)
ret = CPlot.getActiveZones()
print('Active zones: ', ret)
```

CPlot.**getActiveStatus**(*nz*)

Return the active status (1: active, 0: inactive) of zone nz. Active zones are visible, unactive zones are hidden.

Parameters **nz** (int) – number of zone

Returns active status of zone

Return type int

Example of use:

- Get active status of zone (array):

```
# - getActiveStatus (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(5.)
ret = CPlot.getActiveStatus(0)
if ret == 1: print('Zone 0 is active.')
else: print('Zone 0 is inactive.')
```

CPlot.**getActivePoint**()

Return the last clicked point position (coordinates in 3D world) as a list of three coordinates.

Returns active point position

Return type tuple of 3 floats

Example of use:

- Get active point (array):

```
# - getActivePoint (array) -
import Generator as G
import CPlot
import time

a = G.cart( (0,0,0), (1,1,1), (5,5,5) )
CPlot.display([a])

l = []
while l == []:
    l = CPlot.getActivePoint(); time.sleep(0.1)
print('ActivePoint: ', l)
#>> ActivePoint:  [3.9996489035268743, 2.127948294736359, 2.41771355073051]
```

CPlot.**getActivePointIndex**()

Return the active point index. For structured grids, return [ind, indc, i,j,k], where ind is the global index of the nearest node to active point, indc is the global index of the nearest center to active point and i,j,k are the indices of nearest node. For unstructured grids, return [ind, indc, no, 0, 0], where ind is the global index of nearest node, indc is the nearest center to the clicked point and no is the number of ind in the center2node connectivity of indc.

Returns active point index

Return type list of 4 ints

Example of use:

- Get active point index (array):

```
# - getActivePointIndex (array) -
import Generator as G
import CPlot
import time

a = G.cartTetra( (0,0,0), (1,1,1), (5,5,1) )
```

(continues on next page)

(continued from previous page)

```
CPlot.display([a], dim=2)

l = []
while l == []:
    l = CPlot.getActivePointIndex(); time.sleep(0.1)
print('ActivePointIndex : ', l)
#>> ActivePointIndex :  [16, 19, 3, 0, 0]
```

CPlot.**getMouseState**()

Return the current button state of mouse (0: left pressed, 1: middle pressed, 2: right pressed, 5: not pressed) and the current mouse position (if pressed). Use it when dragging.

Returns mouse state and mouse position

Return type tuple of 2 ints

Example of use:

- Get mouse state (array):

```
# - getMouseState (array) -
import Generator as G
import CPlot
import time

a = G.cartTetra( (0,0,0), (1,1,1), (5,5,1) )
CPlot.display([a], dim=2)

c = 1000
while c > 0:
    l = CPlot.getMouseState(); time.sleep(0.5)
    print(l)
    c -= 1
```

CPlot.**getKeyboard**()

Return the pressed keys as a string.

Returns keys pressed in CPlot window

Return type string

Example of use:

- Get keyboard keys (array):

```
# - getKeyboard (array) -
import Generator as G
import CPlot
import Geom as D
import time

a = G.cart((0,0,0), (1,1,1), (8,8,1))
CPlot.display(a, dim=2)
CPlot.setState(activateShortCuts=0)

for i in range(50):
    l = ''
    while l == '':
        l = CPlot.getKeyboard(); time.sleep(0.1)
    l = CPlot.getKeyboard(); CPlot.resetKeyboard()
    try:
        a = D.text2D(l)
        CPlot.display(a)
    except:
        for i in range(len(l)):
            v = ord(l[i]); print(v)
            if v == 1: print('up')
            elif v == 2: print('down')
            elif v == 3: print('left')
            elif v == 4: print('right')
            elif v == 5: print('release up')
            elif v == 6: print('release down')
            elif v == 7: print('release left')
            elif v == 8: print('release right')
        time.sleep(0.1)
        l = ''
```

CPlot.**resetKeyboard**()

Reset the pressed key string stored in plotter.

CPlot.**changeVariable**()

Change displayed variable.

Example of use:

- Change variable (array):

```
# - changeVariable (array) -
import Generator as G
import Converter as C
```

(continues on next page)

(continued from previous page)

```
import CPlot
import time

def F(x,y): return x*x + y*y

a = G.cart((0,0,0), (1,1,1), (5,5,1))
a = C.addVars(a, 'Density')
a = C.initVars(a, 'F', F, ['x','y'])
CPlot.display(a, dim=2, mode=3)

CPlot.changeVariable(); time.sleep(2)
CPlot.changeVariable(); time.sleep(2)
```

CPlot.**changeStyle**()

Change CPlot display style.

Example of use:

- Change Style (array):

```
# - changeStyle (array) -
import Generator as G
import CPlot
import time

a = G.cart((0,0,0), (1,1,1), (5,5,1))
CPlot.display(a, dim=2, mode=1)

CPlot.changeStyle(); time.sleep(2)
CPlot.changeStyle(); time.sleep(2)
```

CPlot.**changeBlanking**()

Change the blanking procedure.

Example of use:

- Change the blanking procedure (array):

```
# - changeBlanking (array) -
import Generator as G
import Converter as C
import CPlot
import time

a = G.cart((0,0,0), (1,1,1), (5,5,1))
```

(continues on next page)

(continued from previous page)

```
a = C.initVars(a, 'cellN', 0)
CPlot.display(a, dim=2, mode=0)

CPlot.changeBlanking(); time.sleep(2)
CPlot.changeBlanking(); time.sleep(2)
```

CPlot.**setState**(*dim*, *mode*, ...)

Set a CPlot state. The same keywords as display can be used.

Additional keywords are:

- **ghostifyDeactivatedZones**: 1 means deactivated zones will appear blended.
- **edgifyActivatedZones**: 1 means activated zones will appear as edges.
- **edgifyDeactivatedZones**: 1 means deactivated zones will appear as edges.
- **message**: "A string" or "Clear"
- **viewAngle**: the camera angle (default: 50 degrees).
- **cursor**: mouse cursor type (0: normal, 1: cross, 2: wait).
- **lightOffset**: offset to default light position (default: (0,0)).
- **dofPower**: power of depth of field effect (default: 3.).
- **gamma**: gamma correction (default: 1.).
- **sobelThreshold**: sobel threshold for zone outlines (default: -0.5).
- **selectionStyle**: style for selection (default: 0).
- **activateShortCut**: if False, deactivate shortCut keys (def: True).
- **billBoards**: list of billboard image files ['file.png',1,1] (default: None).
- **billBoardSize**: size of billboard. If -1, use distance to fit billboards (default: -1).
- **materials**: list of material image files used in textured rendering ['mat.png'] (default: None).

Example of use:

- Set a state in CPlot (array):

```
# - setState (array) -
import Generator as G
import CPlot
```

(continues on next page)

(continued from previous page)

```
import time

a = G.cart((0,0,0), (1,1,1), (5,5,5))
CPlot.display(a, mode='solid')
time.sleep(1.)
CPlot.setState(posCam=(8,8,8), posEye=(5,5,5))
```

CPlot.**setMode**(*mode*)

Set CPlot display mode (0 or 'Mesh': means mesh, 1 or 'Solid': means solid, 2 or 'Render': means render, 3 or 'Scalar' means scalar field, 4 or 'Vector' means vector fields)

Parameters **mode** (int or string) – mode to set

Example of use:

- Set a mode in CPlot (array):

```
# - setMode (array) -
import Generator as G
import CPlot
import time

a = G.cart((0,0,0), (1,1,1), (5,5,1))
CPlot.display(a, mode=0, dim=2); time.sleep(2)
CPlot.setMode(1); time.sleep(2) # solid
CPlot.setMode('mesh'); time.sleep(2)
CPlot.setMode('solid'); time.sleep(2)
```

CPlot.**setSelectedZones**(*list*)

Set the selected zone status (1: selected, 0: not selected) by zone global number.

Parameters **list** (list of tuples of 2 ints) – list of zone number and status

Example of use:

- Set the selected status for a list of zones (array):

```
# - setSelectedZones (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
```

(continues on next page)

(continued from previous page)

```
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(1.)
CPlot.setSelectedZones([(0,1), (1,1)])
```

CPlot.**unselectAllZones**()

Unselect all zones.

Example of use:

- Unselect all zones (array):

```
# - unselectAllZones (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(2.)
CPlot.unselectAllZones()
```

CPlot.**setActiveZones**(*list*)

Set the active (visible) status for given zones.

Parameters **list** (`list of tuples of 2 ints`) – list of zone number and status

Example of use:

- Set the active status for a list of zones (array):

```
# - setActiveZones (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(1.)
CPlot.setActiveZones([(0,0), (1,0)])
```

CPlot.**setZoneNames**(*list*)

Set the specified zone names.

Parameters **list** (list of tuples of int and string) – list of zone number and zone names

Example of use:

- Set the zone names for a list of zones (array):

```
# - setZoneNames (array) -
import Generator as G
import CPlot
import time

a1 = G.cart( (0,0,0), (1,1,1), (5,5,5) )
a2 = G.cart( (7,0,0), (1,1,1), (3,3,3) )
CPlot.display([a1, a2])

time.sleep(1.)
CPlot.setZoneNames([(0,'FirstZone'), (1,'SecondZone')])
```

3.4 Camera setting and motion

CPlot.**lookFor**()

Look for selected zone. It positions the camera for a clear view on the currently selected zone.

Example of use:

- Look for selected zone (array):

```
# - lookFor (array) -
import Generator as G
import CPlot

a = G.cart((0,0,0), (1,1,1), (5,5,5))
CPlot.display(a)
CPlot.lookFor()
```

CPlot.**moveCamera**(*posCams*, *posEyes=None*, *dirCams=None*, *moveEye=False*, *N=100*, *speed=1.*, *pos=-1*)

Move camera along camera points. The list of points specifies the path of the camera. The camera will move along this path, making N steps. pos will position the

camera at step number pos along the path. If posEyes is specified, posEye (that is the position the camera is looking to) will follow this path. If posEyes is not specified and moveEye is true, the posEye will follow the path Otherwise, the posEye will stay at initial posEye.

Parameters

- **posCams** (list of tuple of 3 floats or 1D STRUCT Zone) – coordinates of camera points
- **posEyes** (list of tuple of 3 floats or 1D STRUCT Zone) – coordinates of eyes points
- **dirCams** (list of tuple of 3 floats or 1D STRUCT Zone) – camera directions
- **moveEye** (Boolean) – if True, the eye follow camera points
- **speed** (float) – speed of camera motion
- **N** (int) – number of camera positions
- **pos** (int) – position in posCams (in 0,N)

Returns current posCam, posEye, dirCam

Return type 3 lists of 3 floats

Example of use:

- Move camera along check points (array):

```
# - moveCamera (array) -
import Geom as D
import Converter as C
import Transform as T
import CPlot

# Model
a = D.sphere((0,0,0), 1., N=20)
CPlot.display(a, posCam=(3,-1,0.7), posEye=(0,0,0))

t = 0.
for i in range(1000):
    # change model
    defo = C.initVars(a, '{df}=0.1*cos(%f)*sin(10*pi*{x})'%(t))
    defo = C.extractVars(defo, ['df'])
    b = T.deformNormals(a, defo)
    # Move camera
    CPlot.moveCamera([(3,-1,0.7),(3,5,0.7),(3,7,0.7)], N=1000, pos=i)
    CPlot.display(b)
    t += 0.05
```

- Move camera along check points (pyTree):

```
# - moveCamera (pyTree) -
import Geom.PyTree as D
import CPlot.PyTree as CPlot
import Converter.PyTree as C
import Transform.PyTree as T
import Generator.PyTree as G

# Model
a = D.sphere((0,0,0), 1., N=20)
a = C.convertArray2Hexa(a); a = G.close(a)
CPlot.display(a, posCam=(3,-1,0.7), posEye=(0,0,0))

t = 0.
for i in range(1000):
    # change model
    C._initVars(a, '{df}=0.1*cos(%f)*sin(10*pi*{CoordinateX})'%(t))
    b = T.deformNormals(a, 'df')
    # Move camera
    CPlot.moveCamera([(3,-1,0.7),(3,5,0.7),(3,7,0.7)], N=1000, pos=i)
    CPlot.display(b)
    t += 0.05
```

- Move camera along check points in offscreen mode (pyTree):

```
# - moveCamera (pyTree) -
import Geom.PyTree as D
import CPlot.PyTree as CPlot
import Converter.PyTree as C
import Transform.PyTree as T
import Generator.PyTree as G

# Model
a = D.sphere((0,0,0), 1., N=20)
a = C.convertArray2Hexa(a); a = G.close(a)

t = 0.
for i in range(100):
    # change model
    C._initVars(a, '{df}=0.1*cos(%f)*sin(10*pi*{CoordinateX})'%(t))
    b = T.deformNormals(a, 'df')
    # Move camera
    (posCam, posEye, dirCam) = CPlot.moveCamera([(3,-1,0.7),(3,5,0.7),(3,7,0.
↪7)], N=100, pos=i)
```

(continues on next page)

(continued from previous page)

```
    CPlot.display(b, posCam=posCam, posEye=posEye, dirCam=dirCam, offscreen=1,␣
↪export='image%i.png'%i)
    t += 0.05
```

CPlot.**travelLeft**(*xr*, *N=100*)

Travel camera left/Right/Up/Down/In/Out. Xr is the range (in 0.,1.). N is the number of check points.

Returns final posCam, posEye, dirCam

Return type 3 lists of 3 floats

Example of use:

- Travel camera (array):

```
# - travel* (array) -
import Geom as D
import Generator as G
import CPlot
import Converter as C
import Transform as T

# Model
a = D.sphere((0,0,0), 1., N=20)
a = C.convertArray2Hexa(a); a = G.close(a)
CPlot.display(a, posCam=(3,0,0), posEye=(0,0,0))

t = 0.
for i in range(1300):
    # change model
    defo = C.initVars(a, '{df}=0.1*cos(%f)*sin(10*pi*{x})'%(t))
    defo = C.extractVars(defo, ['df'])
    b = T.deformNormals(a, defo)
    CPlot.display(b)
    t += 0.05

    if i < 200: CPlot.travelLeft(0.001, N=3)
    elif i < 400: CPlot.travelRight(0.001, N=3)
    elif i < 600: CPlot.travelUp(0.001, N=3)
    elif i < 800: CPlot.travelIn(0.001, N=3)
```

- Travel camera (pyTree):

```
# - travel* (array) -
import Geom as D
import Generator as G
import CPlot
import Converter as C
import Transform as T

# Model
a = D.sphere((0,0,0), 1., N=20)
a = C.convertArray2Hexa(a); a = G.close(a)
CPlot.display(a, posCam=(3,0,0), posEye=(0,0,0))

t = 0.
for i in range(1300):
    # change model
    defo = C.initVars(a, '{df}=0.1*cos(%f)*sin(10*pi*{x})'%(t))
    defo = C.extractVars(defo, ['df'])
    b = T.deformNormals(a, defo)
    CPlot.display(b)
    t += 0.05

    if i < 200: CPlot.travelLeft(0.001, N=3)
    elif i < 400: CPlot.travelRight(0.001, N=3)
    elif i < 600: CPlot.travelUp(0.001, N=3)
    elif i < 800: CPlot.travelIn(0.001, N=3)
```

3.5 Set rendering informations in pyTree

CPlot.PyTree.**addRender2Zone**(*a*, *material=None*, *color=None*, *blending=None*, *meshOverlay=None*, *shaderParameters=None*)

Add rendering info to a zone. Info are added in a .RenderInfo user defined node. Use Render mode in display for rendering effects. Exists also as in place version (_addRender2Zone) that modifies a and returns None. Shader parameters are described in *shaderSettings*.

Parameters

- **a** (zone node) – input zone
- **material** (string) – material to set (in 'Solid', 'Flat', 'Glass', 'Chrome', 'Metal', 'Wood', 'Marble', 'Granite', 'Brick', 'XRay', 'Cloud', 'Gooch', 'Sphere')
- **color** (string) – color to set (in 'White', 'Grey', ... or '#FFFF')

- **blending** (float) – opacity factor (in [0.,1.])
- **meshOverlay** (0 or 1) – if 1 then overlay the mesh
- **shaderParameters** (list of two floats in [0.,2.]) – two float that parametrize shaders

Example of use:

- Add render information to zone (pyTree):

```
# - addRender2Zone (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import CPlot.PyTree as CPlot

a = G.cart((0,0,0), (1,1,1), (10,10,1))
C._initVars(a, '{Density}={CoordinateX}')
C._initVars(a, '{centers:VelocityX}={centers:CoordinateY}')
# With a material
a = CPlot.addRender2Zone(a, material='Glass', color='Blue', blending=1.,
                         meshOverlay=1, shaderParameters=[1.,1.])
# With field
a = CPlot.addRender2Zone(a, material='Solid', color='Iso:Density', blending=1.,
                         meshOverlay=1, shaderParameters=[1.,1.])
# With field+material
a = CPlot.addRender2Zone(a, material='Chrome', color='Iso:centers:VelocityX',␣
↪blending=1.,
                         meshOverlay=1, shaderParameters=[1.,1.])

C.convertPyTree2File(a, 'out.cgns')
```

CPlot.PyTree.**addRender2PyTree**(*a*, *slot=0*, *posCam=None*, *posEye=None*, *dirCam=None*, *mode=None*, *scalarField=None*, *niso=None*, *isoScales=None*, *isoEdges=None*, *isoLight=None*, *isoLegend=None*, *colormap=None*, *materials=None*, *bumpMaps=None*, *billBoards=None*)

Add rendering info to a tree. Info are added in a .RenderInfo user defined node. To load the settings to the view, call explicitely CPlot.loadView. Exists also as in place version (_addRender2PyTree) that modifies a and returns None.

Parameters

- **a** (pyTree) – input tree
- **slot** (int) – the slot number
- **posCam** (list of 3 floats) – camera position

- **posEye** (list of 3 floats) – eye position
- **dirCam** (list of 3 floats) – camera direction
- **mode** (string) – displayed mode ('Mesh', 'Solid', 'Scalar', 'Vector', 'Render')
- **scalarField** (string) – scalar field to display in Scalar mode
- **niso** (int) – number of isos to display in Scalar mode
- **isoScales** (list of [string, int, float, float] or [string, int, float, float, float, float]) – list of min and max of a variable ([varName, niso, min, max] or [varName, niso, min, max, cutmin, cutmax])(default: [])
- **isoEdges** (float) – size of edges in Scalar mode
- **isoLight** (0 or 1) – set to 1 if light is used in Scalar mode
- **isoLegend** (0 or 1) – set to 1 if legend is displayed in Scalar mode
- **colormap** (string) – name of the colormap for Scalar mode ('Blue2Red', ...)
- **materials** (list of image file names used for texture mapping) – list of strings
- **bumpMaps** (list of image file names used for bump mapping) – list of strings
- **billboards** – list of strings

Example of use:

- Add render information to pyTree (pyTree):

```
# - addRender2PyTree (pyTree) -
import Converter.PyTree as C
import CPlot.PyTree as CPlot
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,1))
t = C.newPyTree(['Base', a])
# isoScales specify field no, niso for this field, min, max for this field
t = CPlot.addRender2PyTree(t, slot=0, posCam=(1,1,1), posEye=(10,1,1),
                           mode='Solid', niso=10, isoScales=[0, 10, 0., 1.],
                           isoEdges=0.1, isoLight=1, colormap='Blue2Red')
C.convertPyTree2File(t, 'out.cgns')
```

CPlot.PyTree.**loadView**(*a*, *slot=0*)

Load a view defined in a slot to the plotter. A view must already have been stored in pyTree a using CPlot.addRender2PyTree.

Parameters

- **a** (pyTree) – input tree
- **slot** (int) – number of slot to load

Example of use:

- Load (pyTree):

```
# - loadView (pyTree) -
import Converter.PyTree as C
import CPlot.PyTree as CPlot
import Generator.PyTree as G

a = G.cart( (0,0,0), (1,1,1), (10,10,1) )
t = C.newPyTree(['Base', 2, a])

# isoScales specify field no, niso for this field, min, max for this field
t = CPlot.addRender2PyTree(t, slot=0, posCam=(1,1,1), posEye=(10,1,1),
                           mode='Solid', niso=10, isoScales=[0, 10, 0., 1.],
                           isoEdges=0.1, isoLight=1, colormap='Blue2Red')

CPlot.display(t)
import time; time.sleep(0.1)
CPlot.loadView(t, slot=0)
```

3.6 Shader settings

Shaders can be adjusted with the two parameters of the addRender2Zone function.

Here are the meaning of each parameters for each shader:

- Solid: [1] specularFactor; [2] diffuseFactor
- Flat: [1] None; [2] None
- Glass: [1] mix; [2] mix2
- Chrome: [1] mix color/envmap; [2] base color intensity
- Metal: [1] reflection intensity; [2] bump size
- Wood: [1] ray size, [2] bump height

- Marble: [1] ray size, [2] None
- Granite: [1] bump size; [2] bump height
- Brick: [1] brick size; [2] color width
- XRay: [1] fall off; [2] color modifier
- Cloud: [1] size; [2] None
- Gooch : [1] width; [2] shininess
- Sphere: [1] size of sphere; [2] type of billboard
- TexMat: [1] specularFactor; [2] texture number

CHAPTER

FOUR

INDEX

- genindex
- modindex
- search