CGAL PYTHON Visualization

Μιχαήλ Νικολάου

Expanding the current version!

cgVisual class

- Only one instance: cgVis, contains global information
- Currently for one vpython window

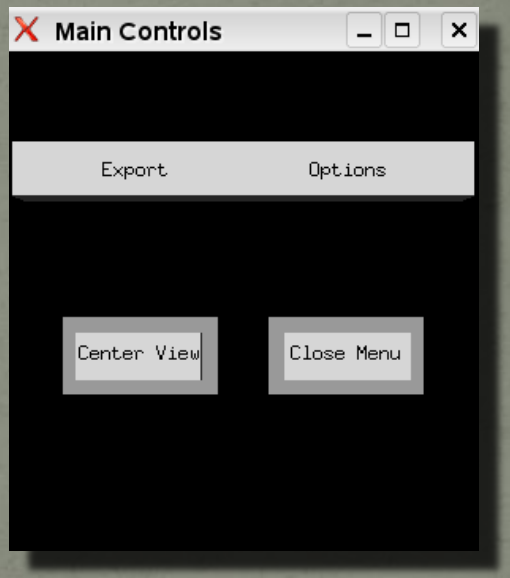
Controls

- Controls are done using the vpython controls module
- So this means that we have do deal with anything good or bad that the module has
- In the future, we could use something more versatile like tcl/tk?
- Use prefined vpython controls, that include buttons, toggles etc.

Controls – Main Controls

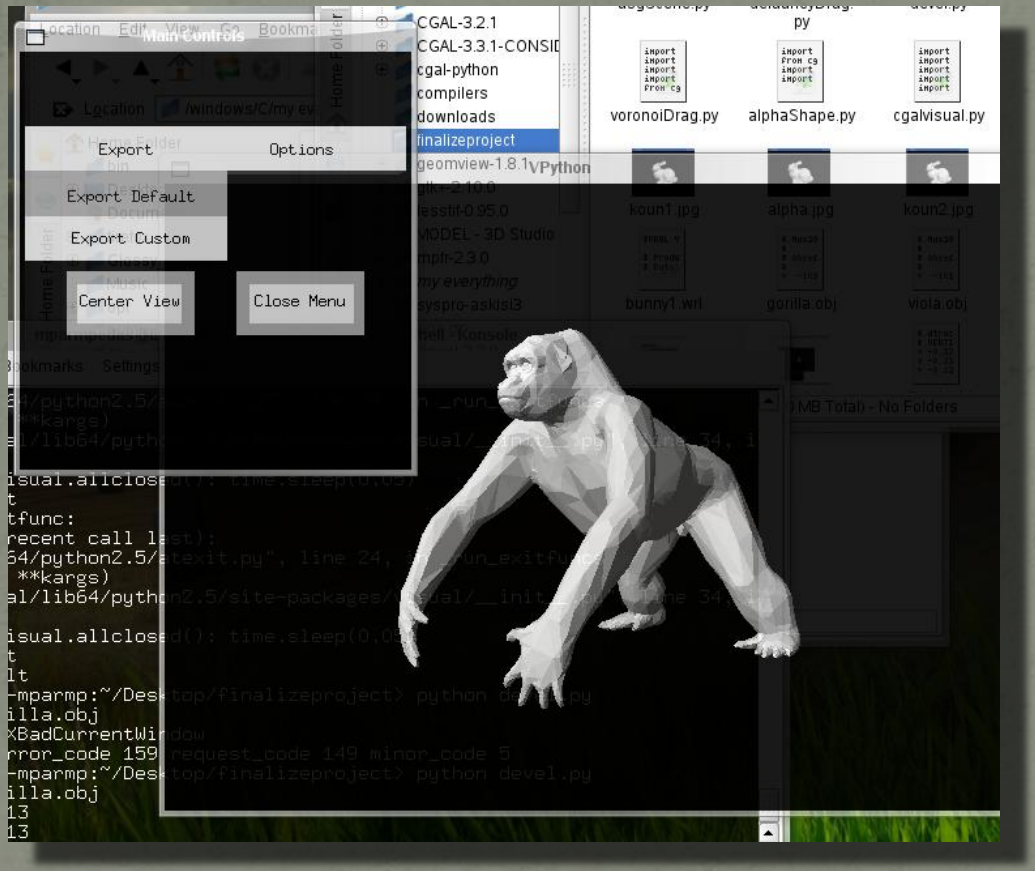
- Start the main controls window calling cgVis.mainControls(True)
- Close the main controls window calling cgVis.mainControls(False)
- Runs in a different thread to avoid having the user call interact() all the time

- Through the main controls you can:
  - Export the scene into an ascii file
    (so that it can be imported later)
  - Change the point ratio (smaller or bigger)
  - Change the default export file
  - Center the visual scene



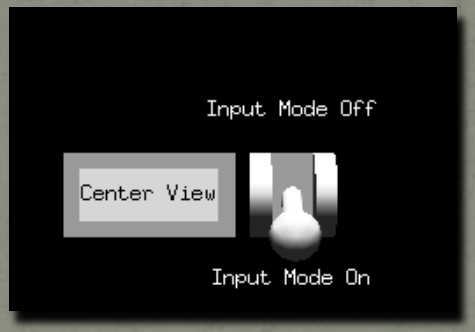
Controls – Main Controls

…



Controls – Input Controls

- Used on 3d Input (because right click terminates input on 2d) to terminate input mode
- Destroys itself when input is done
  - Centers a 3d input screen
  - Turns input mode off



Handling Scenes

- use cgVis.exportScene(filename) to export the scene
- cgVis.importScene(filename) to import the scene
- String representations (repr(...)) of objects are saved in the ascii file.
- Uses eval() to evaluate the repr
- Useful for reproducing conditions such as interesting instances, faulty instances

Handling Scenes
- how does the output file look like?

- VSegment_2(CGAL.Point_2(5.94333290425,-5.34941304484), CGAL.Point_2(10.0302093094,-9.23989525928), color = (1, 1, 1), radius = 0)
- VPoint_3(-10.641452494699999, -16.206391615200001, 11.995322014999999, color = (1, 1, 1), label = None, radius = 0.5)
- VPoint_3(-5.68106430892999999, -1.4589308304099999, 17.0876748188, color = (1, 1, 1), label = None, radius = 0.5)
- VSegment_2(CGAL.Point_2(15.3567496976,-16.0482391345), CGAL.Point_2(11.3709628331,-19.6955662106), color = (1, 1, 1), radius = 0)
- VPoint_3(10.5207495504, 13.530106761300001, -10.9352694549, color = (1, 1, 1), label = None, radius = 0.5)
- VPoint_2(-8.25434233758875546, 7.4162317212614788, color = (1, 1, 1), label = None, radius = 0.5)
- VPoint_3(-4.4919450961000003, -0.60788784600500001, 12.342158570600001, color = (1, 1, 1), label = None, radius = 0.5)

Handling Scenes
- how does the output file look like?

- So, calling eval(VPoint_3(-10.6, -16.2, 11.9, color = (1, 1, 1), label = None, radius = 0.5)) opens up a vpython window and outputs the point with coordinates (-10.6, -16.2, 11.9)

Handling Scenes

How is this done?

- cgVis.cgalSceneReg is a dictionary that holds all visual objects created (see Vbase constructor)
- The key is the id of the object and the value is the reference to it.
- So if v is a VPoint_2, it is stored as cgVis.cgalSceneReg[id(v)] = v
- What happens with garbage collection?
- VPython`s philosophy is to hold a reference to any visual object, so even if no reference exists from the user the object remains visible. Only by turning the visual object invisible it is garbage collected
- This is maintained, because whenever we turn an object invisible it is popped of the dictionary, so can be garbage collected! (see VBase. __setVisible for more)

Handling Scenes

- Related usage files:
  - usgScene.py

3d Models (Wavefront .obj)

- Loading 3d models into the visual module
- Compatible with cgal-python classes
- obj3dLoader class
- Actually, a parser. Loads wavefront points (vertices) and faces (triangles) into Point_3 and Triangle_3

3d Models (Wavefront, .obj)

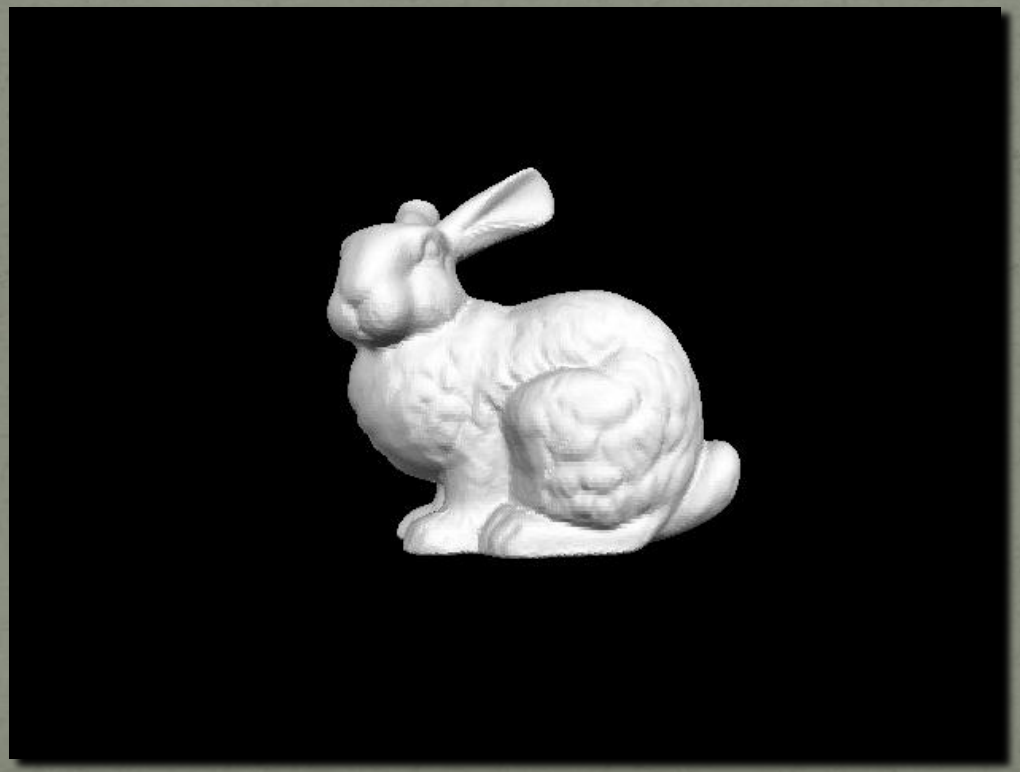
- Usage:
  - c = obj3dLoader("3dModels/bunny.obj", 200)
  - First argument is the 3d model, second argument is the scaling factor (multiplies each coordinate by that amount)
  - Class constructor parses the text file and then centers the 3d object (calling the centerObj method)
  - Can call scale() to rescale
  - draw_Triangles() draws the faces and draw_Points() draws the points (take care of the radius)
  - The class can be easily expanded to adding translation functions etc.

3d Models

- Public Members:
  - .vertices: CGAL.Point_2 points
  - .faces: Pointers to vertices (index in list)
  - .vfaces: VTriangle_3 (if draw_Triangles was called in the past)
  - .vvertices: VPoint_3 (if draw_Points method was called in the past)

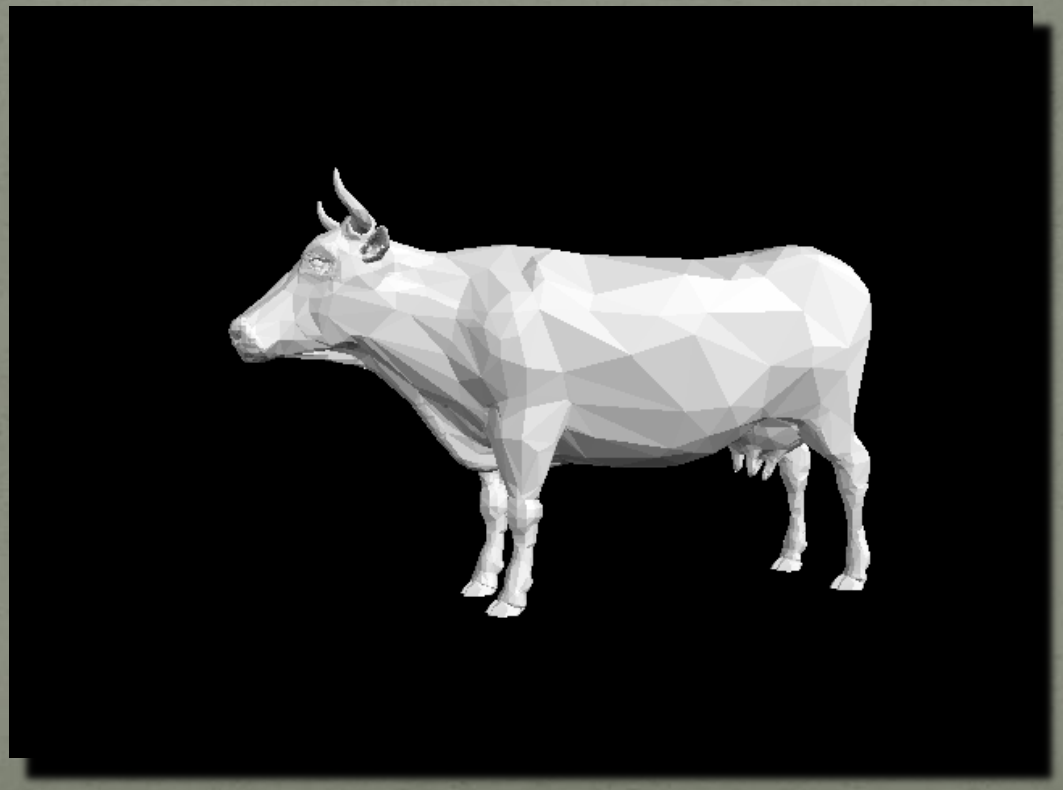
3d Models - examples

- c = obj3dLoader("3dModels/bunny.obj", 200)
- c.draw_Triangles()



3d Models - examples

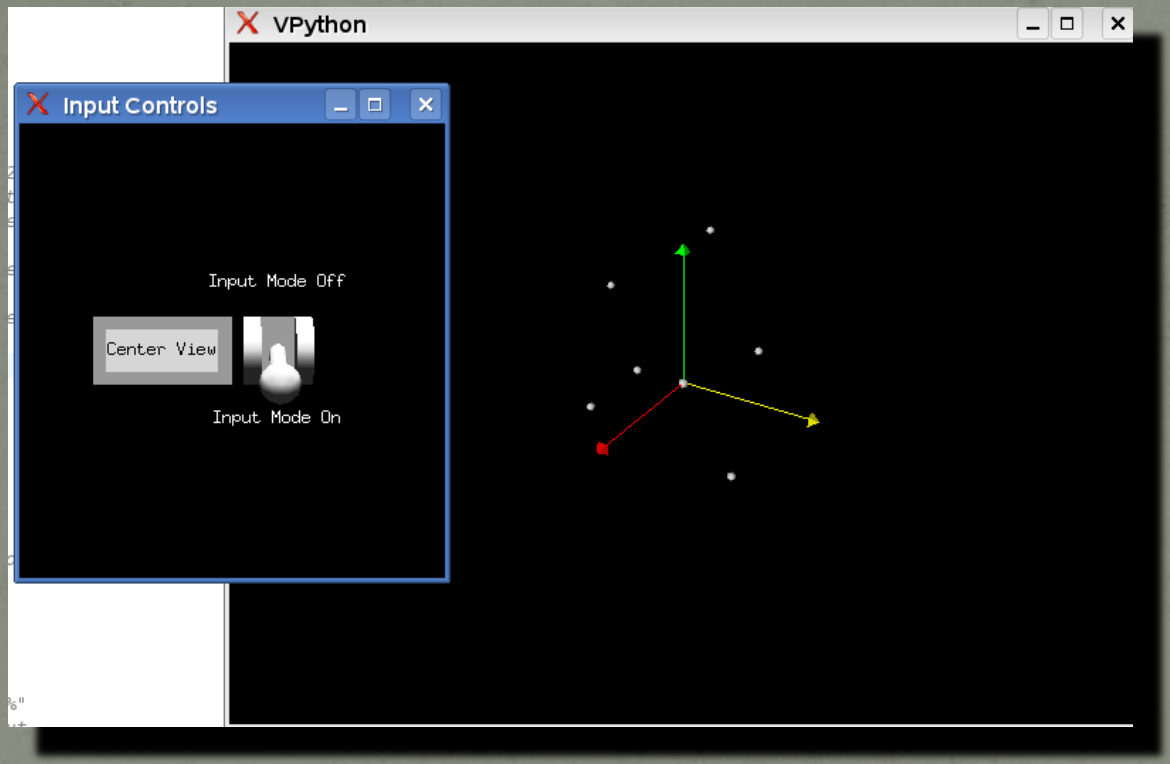
- c = obj3dLoader("3dModels/cow.obj", 0.06)
- c.draw_Triangles()



Input

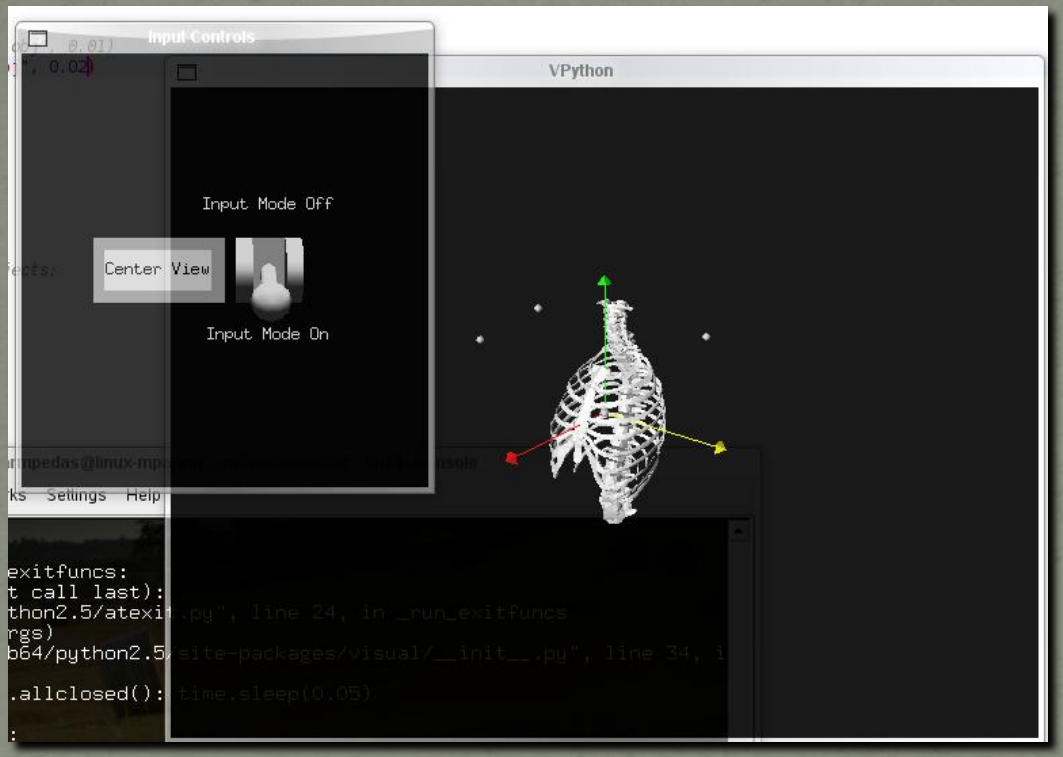
- All input (2d & 3d) is now click n drag enabled
- getVisualPoints_2(), getVisualPoints_3()
- getPolygon_2(), getPolygon_3()

3d Input


VPython
Input Controls
Input Mode Off
Center View
Input Mode On

3d Input – 3d Objects

- Combining 3d input and a 3d model

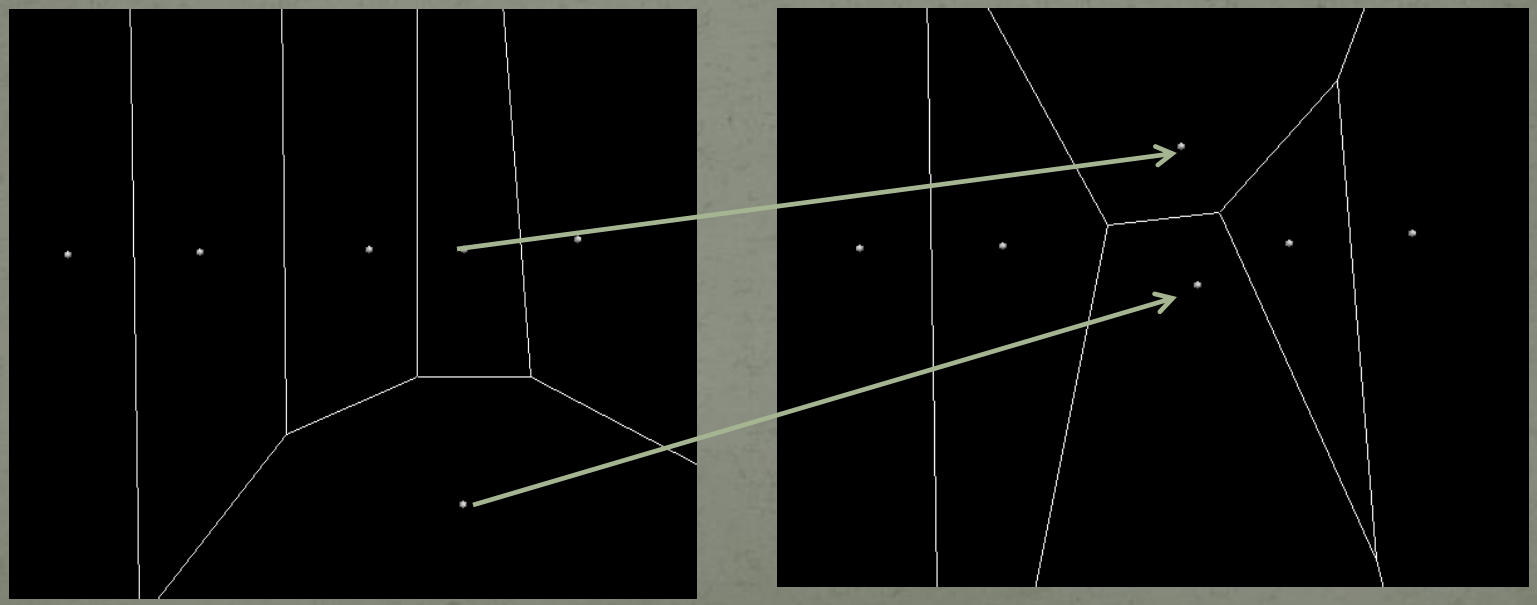


The clicknDrag class

- Highly customizable
- Iterates and catches user clicks, drags
- By default, loops while cgVis.inputMode = True
- Arguments are quite self-explanatory, input functions tell the class what to do when a user drags a point, clicks on a point, creates a point (if user allows it) or drops a point

The clicknDrag class

- Usage Files:
  - voronoiDrag.py
  - Draws the voronoi diagram, with draggable sites: The user can drag the sites and observe the changes that occur to the voronoi diagram



The clicknDrag class

- Usage Files:
  - voronoiDrag.py
  - cnd = clicknDrag(points,_3d = False, doWhenNewPoint=updateVor, doWhenDropped=updateVor,doWhileDragged=updateVor, terminateCondition=None, terminateOnRightClick=True)

  - This initializes the clicknDrag method for the voronoiDrag script. The input seems a bit long, but that is because the names of the arguments are actually explaining themselves

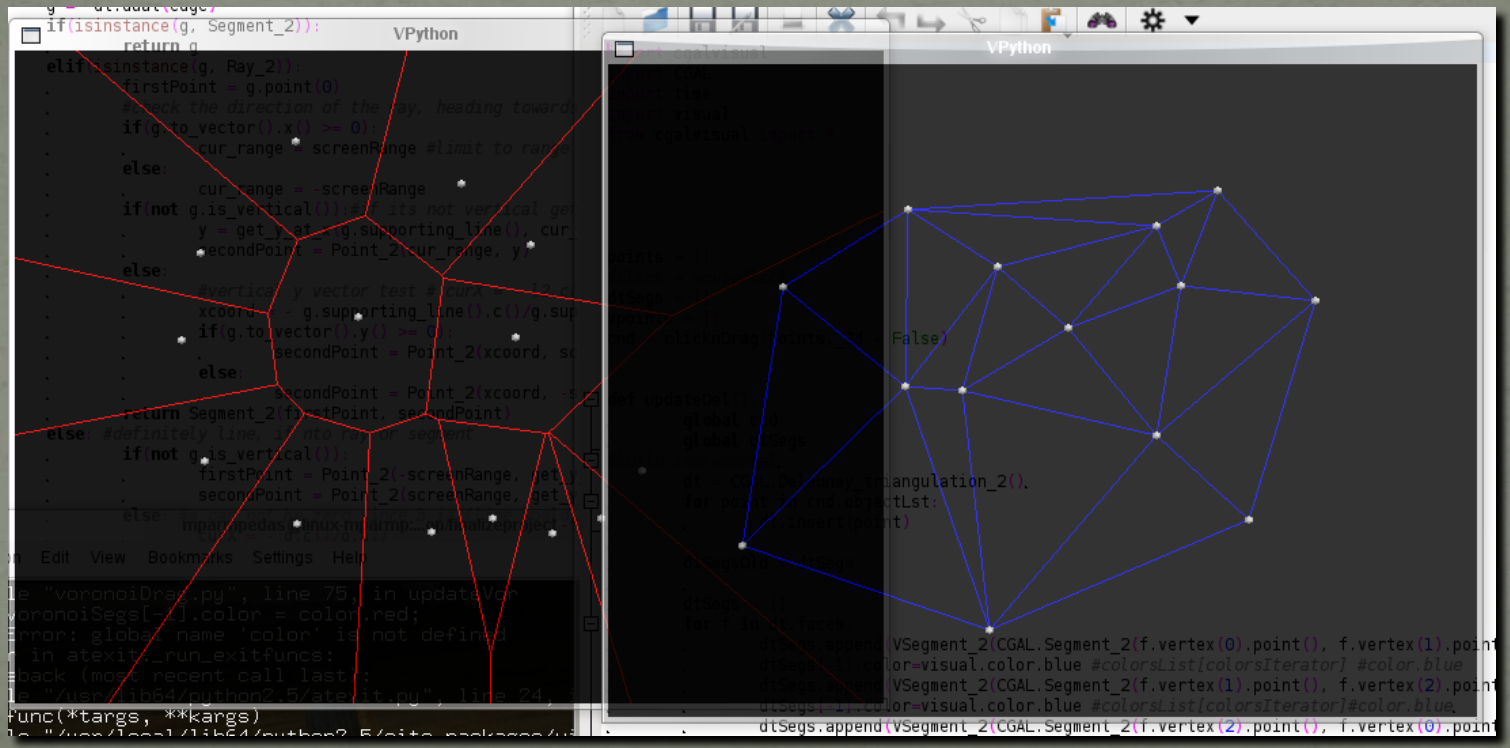
The clicknDrag class

- Usage Files:
  - voronoiDrag.py
  - cnd = clicknDrag(points,_3d = False, doWhenNewPoint=updateVor, doWhenDropped=updateVor,doWhileDragged=updateVor, terminateCondition=None, terminateOnRightClick=True)

  - As input, a list of points is given, the input is 2d (so 3d = False), and the function updateVor is called when a new point is created – user clicks, while a point is dragged and when a point is dropped.
  - The function loops, until user right clicks! (terminate OnRightClick = True)

The clicknDrag class

- Usage Files:
  - delauneyDrag.py – Does the same thing but presents the delauney triangulation

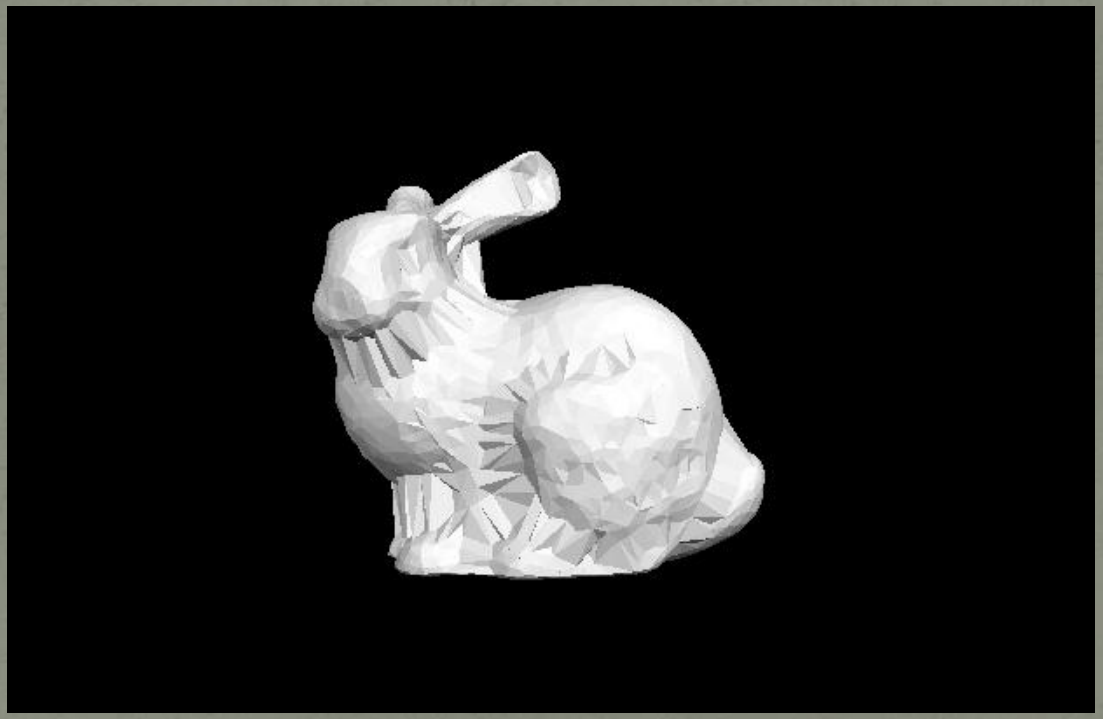


Alpha Shapes

- File: alphaShape.py
- Use the visual library to visualize the alpha shape of a 3d model

Alpha Shapes

- Bunny



Alpha Shapes

- Gorilla

Misc

- Line_2 repr() seems to be buggy in the CGAL module
- We do not import CGAL module members (we use import CGAL and not from CGAL import *), so we are not “inside” the CGAL namespace. So we need to append CGAL. to visual object representations (string formatting)
- Added color to repr() of objects such as Triangle_2, Triangle_3 so that the user can save color on exporting the scene
- Could have more scene parameters as methods, such as default point radius, rotation on / off etc.
- Little time, many things to add and improve, probably a few bugs;p