Introduction to Python and VTK

Scientific Visualization, HT 2014
Lecture 2

Johan Nysjö
Centre for Image analysis
Swedish University of Agricultural Sciences
Uppsala University
About me

- PhD student in Computerized Image Analysis
- Develop methods and tools for interactive analysis of medical 3D (volume) images
History

- The Python programming language was developed in the late 1980s by a Dutch computer programmer named Guido Van Rossum (who now is the Benevolent Dictator for Life of the language)
- First version released in 1991
- Named after the Monty Python comedy group, not the snake...
Key features

- General-purpose, high-level programming language
- Clear, readable syntax (similar to pseudocode)
- Dynamically AND strongly typed (see explanation here)
- Multi-paradigm: you can write code that is (fully or partially) procedural, object-oriented, or functional
- No compiling*
- Has extensive standard libraries and a rich selection of third-party modules
- Good for rapid prototyping

* some compiling is performed in the background, but at least you don’t have to think about it
Running a Python program

• Suppose that we have a program hello.py containing this single line of code:

```python
print("Hello world!")
```

• To run this program, just open a terminal, navigate to the directory of the file, and type

```
johan@hastur:~$ python hello.py
Hello world!
johan@hastur:~$
```
Built-in numeric types

- Integers (int): 1, 2, 3
- Floats (float): 0.1, 3.141592 (64-bit by default)
- Complex: 0+1j, 1.1+3.5j
- Booleans: True, False
Container types

- Strings (str): "python", "foo"
- Lists (list): [1, 2, 3], [0.5, "bar", True], [[0, 1, 0], [1, 0, 0]]
- Tuples (tuple): (1, 2, 3)
- Dictionaries (dict): {"key0": 1.5, "key1": 3.0}

- Strings and tuples are immutable (i.e., cannot be modified after creation), whereas lists and dictionaries are mutable (can be modified)

- Lists, tuples and dictionaries can contain mixed types
Control flow

- No switch-statement, but otherwise all the familiar control-flow statements are there. Examples:

```python
numbers = [0, 1, 2, 3, 4]
for number in numbers:
    print(number)

for i in xrange(0, len(numbers)):
    print(numbers[i])

i = 0
while i < 10:
    print(i)
    i += 1
```
Functions

- Functions are defined like this:

```python
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

if __name__ == "__main__":
    print(fibonacci(10))
```
Whitespace-sensitive syntax

- Python uses ":" and whitespace indentation to delimit code blocks, e.g., define where a function or control-flow statement starts and stops
- Controversial design choice...
- Forces you to write readable (or at least well-indented) code

```python
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

if __name__ == "__main__":
    print(fibonacci(10))
```
File I/O

• Using the `with` statement (available since Python 2.5), reading or writing to file is really simple:

```python
# reading from file
with open("data.txt", "r") as txt_file:
    content = txt_file.read()

# writing to file
with open("output.txt", "w") as txt_file:
    txt_file.write("Some data")
```
Classes

- Python supports object-oriented programming

```python
import math

class Sphere:
    def __init__(self, center=[0.0, 0.0, 0.0], radius=1.0):
        self.center = center
        self.radius = radius

    def compute_volume(self):
        return (4.0 / 3.0) * math.pi * math.pow(self.radius, 3)

sphere = Sphere()
print(sphere.compute_volume())
```

(unlike Java or C++, getters and setters are normally not used in Python)
Modules

- Every *.py file is a module
- Related functions and classes should be grouped into modules
- You can then use the `import` statement to import the module (or some selected part of it) into your script
- Related modules can be grouped into a package (good if you plan to distribute your code)
The Python standard library

- Provides modules for file and directory access, mathematics, testing, GUI programming, networking, etc
- Read more about it on http://docs.python.org/2/library/index.html
- Some useful modules from the standard library are
  - **math** (mathematical functions and constants)
  - **os** (operating system functionality)
  - **sys** (system-specific parameters and functions)
Python versions (2.x vs. 3.x)

- The Python 3.x branch is a revision of the language and offers many improvements over Python 2.x
- However, Python 3.x is not backward-compatible, and many existing packages (e.g., VTK) for Python 2.x have not yet been ported to Python 3.x
- Python 2.x is still more widely used
- See http://wiki.python.org/moin/Python2orPython3 for more info
- In this course we will use Python 2.6 or 2.7
Text editors, IDEs, and interactive shells

- To start programming in Python, almost any text editor or IDE with Python support will do.
- The standard Python shell is great for trying out language features.
- For a more powerful interactive computing environment, have a look at IPython.
Style guide for Python code (PEP8)

- To simplify the life for Python programmers, some of the language developers sat down and wrote a style guide for Python code: PEP8
- The guidelines in PEP8 are just recommendations: you are free to break them and define your own coding style guide (but please be consistent)
When you need more speed

- NumPy & SciPy
- Cython (supports parallel processing via OpenMP)
- PyCUDA
- PyOpenCL
Other useful packages

- Graphics programming and visualization
  - PyOpenGL, VTK, Mayavi
- GUI programming
  - PyQt/PySide, wxPython, Tkinter
- Image analysis and processing
  - ITK, Pillow
- Computer vision
  - OpenCV
- Plotting
  - Matplotlib
Python tutorials

• If you are new to Python, start with:
  https://docs.python.org/2/tutorial/

• Zed Shaw's "Learning Python The Hard Way" is also a good (but more demanding) tutorial:
  http://learnpythononthehardway.org/book/
The Visualization Toolkit (VTK)

- Open source, freely available C++ toolkit for
  - scientific visualization
  - 3D computer graphics
  - mesh and image processing
- Managed by Kitware Inc.
VTK

- Object-oriented design
- High level of abstraction (compared to graphics APIs like OpenGL or Direct3D)
- Provides bindings to Tcl/Tk, Python, and Java
- GUI bindings: Qt, wxWidgets, Tkinter, etc
Heavily object-oriented (and a bit over-designed...)
Some examples of what you can do with VTK

- Create visualizations of
  - scalar, vector, and tensor fields
  - volume data (e.g., 3D CT or MRI scans)
- Mesh and polygon processing
- Image analysis (2D and 3D images)
- Isosurface extraction
- Implementing your own algorithms
Volume rendering
Rendering graphical 3D models
(imported from .stl, .ply, .obj, etc)
Rendering performance

- VTK has decent rendering performance and is good for rapid prototyping of 3D visualization tools
- Not suitable for rendering large realistic 3D scenes with lots of dynamic content (i.e., games)
The visualization pipeline

# vtk DataFile Version 3.0
vtk output
BINARY
DATASET STRUCTURED_POINTS
DIMENSIONS 256 256 124
SPACING 0.9 0.9 0.9
ORIGIN 0 0 0
CELL_DATA 7998075
POINT_DATA 8126464
COLOR_SCALARS ImageFile 1
^D^E^C^G^D^B^D^B^B^C^D^E^D^E^C^C
^D^C^C^C^C^E^D^C^A^B^B^B^F^A^C^E
^E^D^E^A^A^C^B^B^E^B^A^A^E^B^E^E
^A^C^C^G^C^D^F^B^D^E^@^G^C^D^D^C
^D^C^F^C^B^E^E^E^B^C^C^B^C^B^C^B
^C^C^F^E^F^C^D^A^A^C^F^D^D^E^E^B
The visualization pipeline

- To visualize your data in VTK, you normally set up a pipeline like this:
Sources

- VTK provides various source classes that can be used to construct simple geometric objects like spheres, cubes, cones, cylinders, etc...
- Examples: `vtkSphereSource`, `vtkCubeSource`, `vtkConeSource`

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Readers

- Reads data from file
- You can use, e.g., `vtkStructuredPointsReader` to read a volumetric image from a .vtk file
- or `vtkSTLReader` to load a 3D polygon model from a .stl file
- If VTK cannot read your data, write your own reader!

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Filters

- Takes data as input, modifies it in some way, and returns the modified data.
- Can be used to (for example):
  - select data of a particular size, strength, intensity, etc.
  - process 2D/3D images or polygon meshes.
  - generate geometric objects from data.

source/reader → \textcolor{red}{filter} → mapper → actor → renderer → renderWindow → interactor
Mappers

- Maps data to graphics primitives (points, lines, or triangles) that can be displayed by the renderer.
- The mapper you will use most in the labs is `vtkPolyDataMapper`.
- `vtkPolyDataMapper` maps polygonal data (`vtkPolyData`) to graphics primitives.

source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Image source: http://www.realtimerendering.com
Actors

- **vtkActor** represents an object (geometry and properties) in a rendering scene.
- Has position, scale, orientation, various rendering properties, textures, etc. Keeps a reference to the mapper.

source/reader → filter → mapper → **actor** → renderer → renderWindow → interactor
Rendering

- The process of converting 3D graphics primitives (points, lines, triangles, etc), a specification for lights and materials, and a camera view into an 2D image that can be displayed on the screen
**Renderer**

- *vtkRenderer* controls the rendering process for actors and scenes
- Under the hood, VTK uses OpenGL for rendering

source/reader → filter → mapper → actor → **renderer** → renderWindow → interactor

Image source: http://www.realtimerendering.com
The `vtkRenderWindow` class creates a window for renderers to draw into.

```
source/reader → filter → mapper → actor → renderer → renderWindow → interactor
```
Interactors

- The `vtkRenderWindowInteractor` class provides platform-independent window interaction via the mouse and keyboard.
- Allows you to rotate/zoom/pan the camera, select and manipulate actors, etc.
- Also handles time events.

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Example 1:
Rendering a cube
Pipeline for the cube example

1. `vtkCubeSource`
2. `vtkPolyDataMapper`
3. `vtkActor`
4. `vtkRenderer`
5. `vtkRenderWindow`
6. `vtkRenderWindowInteractor`
import vtk

# Generate polygon data for a cube
cube = vtk.vtkCubeSource()
# Create a mapper for the cube data

cube_mapper = vtk.vtkPolyDataMapper()
cube_mapper.SetInput(cube.GetOutput())

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Actor

```python
# Connect the mapper to an actor
cube_actor = vtk.vtkActor()
cube_actor.SetMapper(cube_mapper)
cube_actor.GetProperty().SetColor(1.0, 0.0, 0.0)
```

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
# Create a renderer and add the cube actor to it
renderer = vtk.vtkRenderer()
renderer.SetBackground(0.0, 0.0, 0.0)
renderer.AddActor(cube_actor)

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
# Create a render window
render_window = vtk.vtkRenderWindow()
render_window.SetWindowName("Simple VTK scene")
render_window.SetSize(400, 400)
render_window.AddRenderer(renderer)

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
# Create an interactor
interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)

# Initialize the interactor and start the
# rendering loop
interactor.Initialize()
render_window.Render()
interactor.Start()
Source code – cube.py

• Included in the .ZIP file containing the source code and datasets for Lab 1
• You can download it from the course webpage
Example 2: Earthquake data

1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Visualizing the quakes with sphere glyphs

1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Sphere glyphs

Strength
- vtkFloatArray
  - vtkPolyData

Position
- vtkPoints
  - vtkSphereSource
    - vtkGlyph3D
      - vtkPolyDataMapper
        - vtkActor
      - vtkColorTransferFunction
See this paper for a discussion on why the "rainbow" colormap is a poor choice for most applications.
Colormaps

1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Colormaps

1: Change colormap  
2: Toggle map  
4: Toggle grid  
5: Toggle heightmap  
6: Toggle outline  
7: Start animation
Example 2: Air currents
Arrow glyphs, first try
Arrow glyphs, first try

Direction and speed

- `vtkStructuredPoints`
- `vtkArrowSource`

- `vtkGlyph3D`
- `vtkColorTransferFunction`

- `vtkPolyDataMapper`
- `vtkActor`
Cut planes
Cut planes

Direction and speed

- vtkStructuredPoints
- vtkImageDataGeometryFilter
- vtkPolyDataMapper
- vtkActor
- vtkColorTransferFunction
Arrow glyphs, second try
Arrow glyphs, second try

Direction and speed

- vtkStructuredPoints
- vtkImageDataGeometryFilter
- vtkArrowSource
- vtkGlyph3D
- vtkColorTransferFunction
- vtkPolyDataMapper
- vtkActor
Streamtubes

Direction and speed
- vtkStructuredPoints

Seeds (starting points)
- vtkPointSource

vtkStreamLine

vtkTubeFilter

vtkColorTransferFunction

vtkPolyDataMapper

vtkActor
Example 3: Medical 3D data
Outline

1: Toggle MPR
2: Toggle segmentation
Multi-planar reformatting (MPR)
Multi-planar reformatting (MPR)
Surface rendering
Surface rendering

- Segmented volume image
  - `vtkStructuredPoints`
  - `vtkImageGaussianSmooth`
  - `vtkContourFilter`
  - `vtkPolyDataMapper`
  - `vtkActor`
Combined visualization

1: Toggle MPR
2: Toggle segmentation
Summary

- VTK contains thousands of classes and might seem a bit intimidating at first...
  - however, one can create useful visualizations with just a few core classes
- The pipeline is typically
  
  source/reader → filter → mapper → actor → renderer
  → renderWindow → interactor
- Use VTK's example programs as templates when you write new programs!
Resources

- http://www.vtk.org/
- http://www.vtk.org/VTK/resources/software.html
- http://www.vtk.org/Wiki/VTK/Examples
More resources

- Anders has created a tutorial demonstrating how to use VTK with Python
- Includes lots of examples
- You can access the tutorial here
About the labs

- There will be two assignments and one project
- The lab sessions will be in PC-lab 1312 and 1313
- You may work individually or in pairs
- VTK is installed on the lab PCs
- We recommend that you also install Python and VTK on your own computer
Installing VTK on Linux

- Included in the package repository of most Linux distributions
- On Ubuntu 12.04 you can install VTK and the Python-wraper with the command
  
  `sudo apt-get install libvtk5-dev python-vtk`

- Also fairly easy to build VTK from source. You need GCC, CMake, + some extra dependencies
- Finally, you can install VTK via the Python distribution Anaconda (see next slide)
Installing VTK on Windows

- Don't bother compiling it yourself (unless you have plenty of time to spare)
- Install it via one of the following Python distributions:
  - Anaconda (VTK is available in the package repository)
  - pythonxy (Warning! will override existing Python installations)
- More detailed installation instructions can be found on the course webpage
Installing VTK on Mac

- Install it via Anaconda (see previous slide)
- Expect to spend several hours in front of the compiler if you try to build it yourself...
Paraview and Mayavi

- Free data visualizers built on VTK
- You can use them to try out different visualization techniques (without writing a single line of code)
- Links:
  - http://www.paraview.org/
See you on the lab tomorrow!