The Visualization Pipeline

CS9223 Data Visualization
NYU-Poly

Outline

- Procedural vs. Dataflow Programming
- Dataflow for Visualization Pipeline
- VTK Pipeline
- Quick guide on CMake/VTK and VisTrails

Programming Paradigms

- Imperative/Structured
  - Procedural
  - Object-oriented
- Declarative
  - Dataflow
  - Functional
- ... many many more (check out Wikipedia)

Programming Paradigms

- Imperative/Structured
  - Procedural
  - Object-oriented
- Declarative
  - Dataflow
  - Functional
- ... many many more (check out Wikipedia)

Explicitly describe every step of execution, focus on the computation.

Programming Paradigms

- Imperative/Structured
  - Procedural
  - Object-oriented
- Declarative
  - Dataflow
  - Functional
- ... many many more (check out Wikipedia)

Focus on how computations are connected, graphical specification of procedures.

Example

Given a program implementation:

```
int A() {...}
int B() {...}
int C(int,int) {...}
```

Example

Given a program implementation:

```
int A() {...}
int B() {...}
int C(int,int) {...}
```

Procedural programming:

```
outA = A();
outB = B();
outC = C(outA,outB);
```

Example

Given a program implementation:

```
int A() {...}
int B() {...}
int C(int,int) {...}
```

Procedural programming:

```
outA = A();
outB = B();
outC = C(outA,outB);
```

The order of execution: A→B→C

NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

Example

Given a program implementation:

```
int A() {...}
int B() {...}
int C(int,int) {...}
```

Procedural programming:

```
outB = B();
outA = A();
outC = C(outA,outB);
```

The order of execution: B→A→C

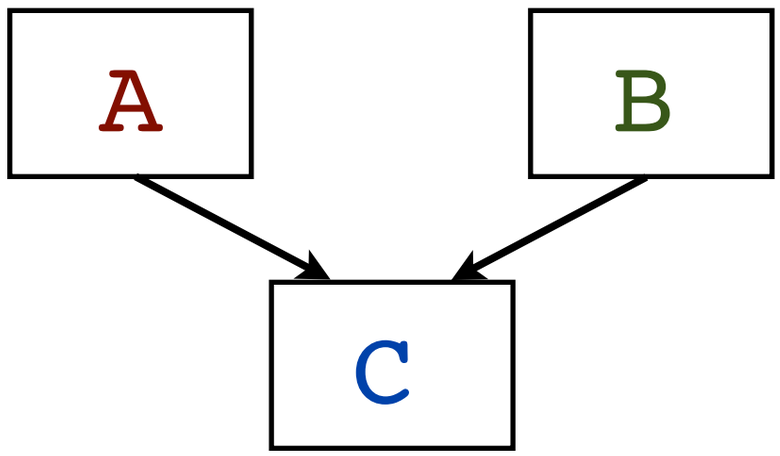
NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

Example

Given a program implementation:

```
int A() {...}
int B() {...}
int C(int,int) {...}
```

Dataflow Programming:

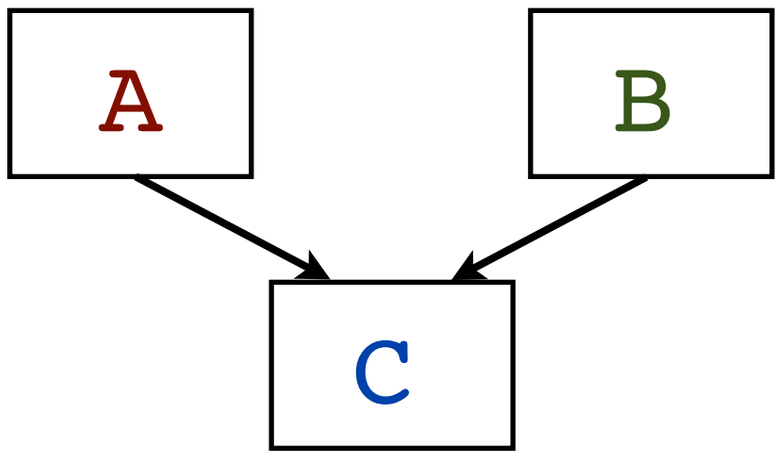


Example

Given a program implementation:

```
int A() {...}
int B() {...}
int C(int,int) {...}
```

Dataflow Programming:



The order of execution: (A∪B)→C

Procedural vs. Dataflow Programming

- Procedural Programming:
  - The control flow has to be dictated
  - Sequential execution
  - Difficult to manage in large execution networks
- Dataflow:
  - The control flow is specified through data dependency
  - Freedom in execution order → concurrency
  - More flexible and easier for use

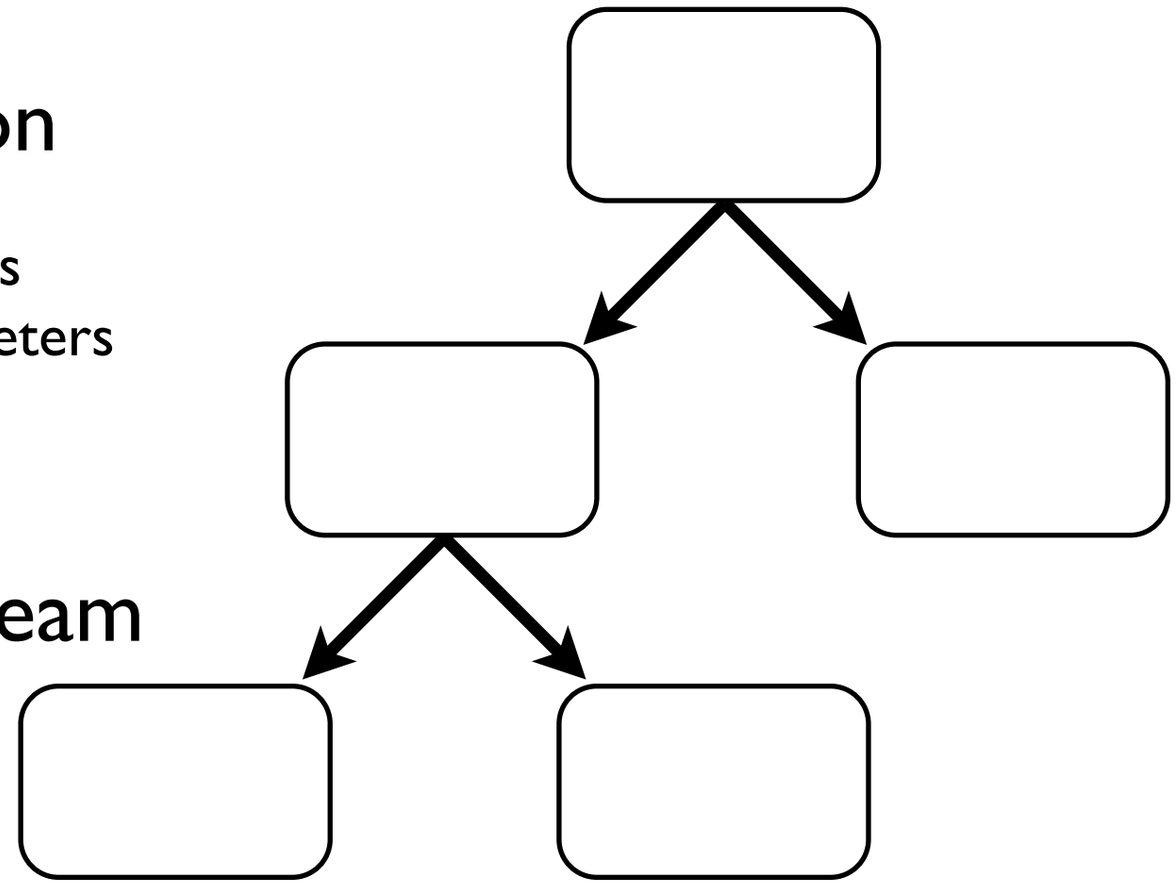
Dataflow Construction

- A directed graph
  - node (module) = computation

knows how to compute outputs
given a set of inputs and parameters

- edge (connection) = data stream

maintains data states and specifications

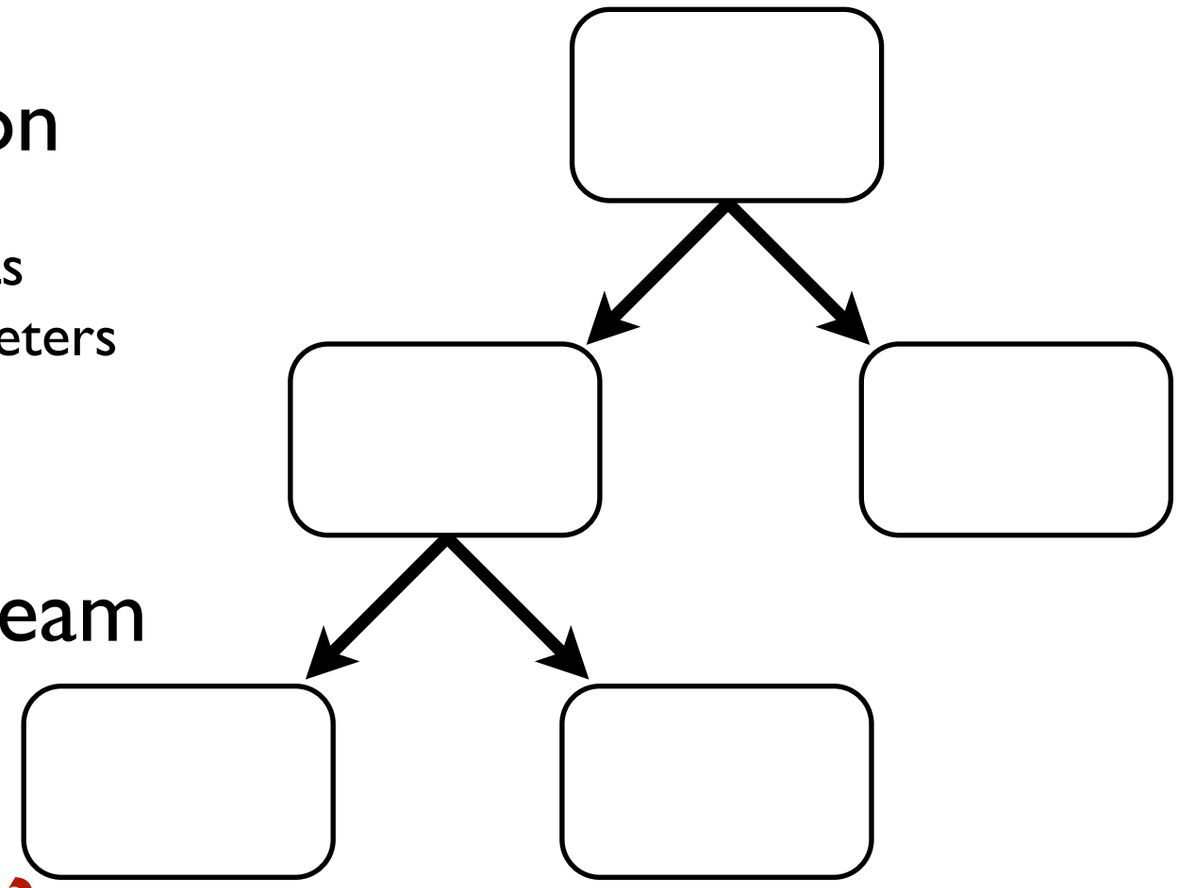


NYU:poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

Dataflow Construction

- A directed graph
  - node (module) = computation

knows how to compute outputs
given a set of inputs and parameters



  - edge (connection) = data stream

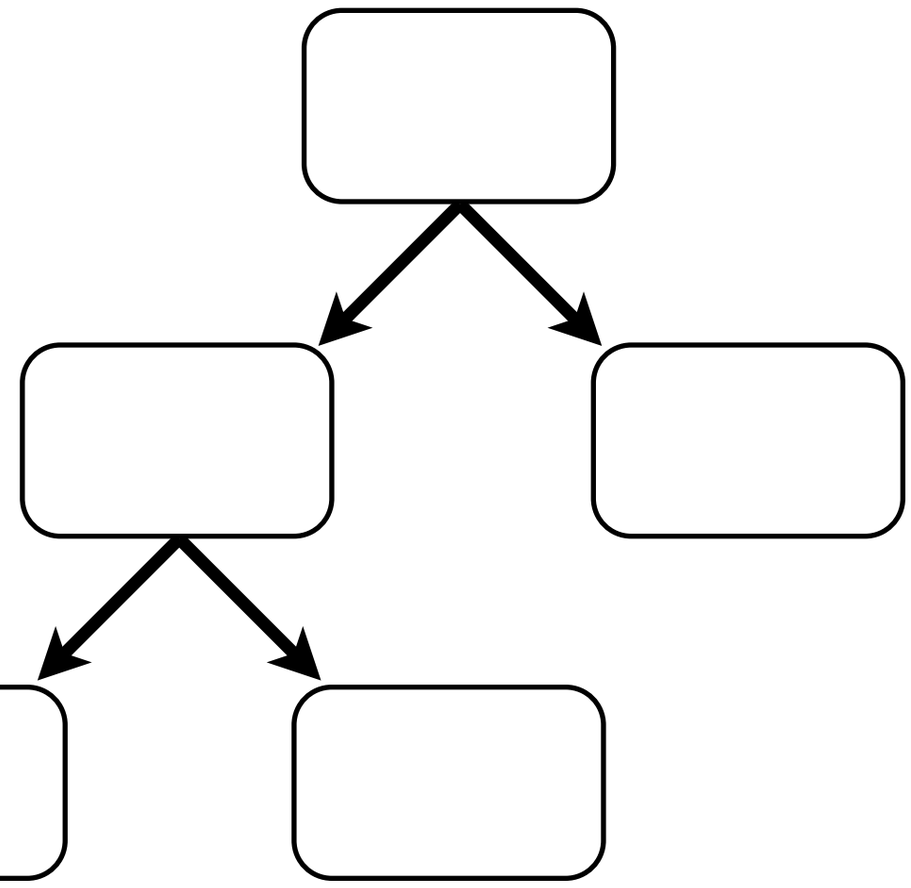
maintains data states and specifications

What controls the execution?

Dataflow Construction

- A directed graph
  - node (module) = computation

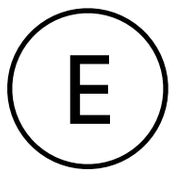
knows how to compute outputs
given a set of inputs and parameters



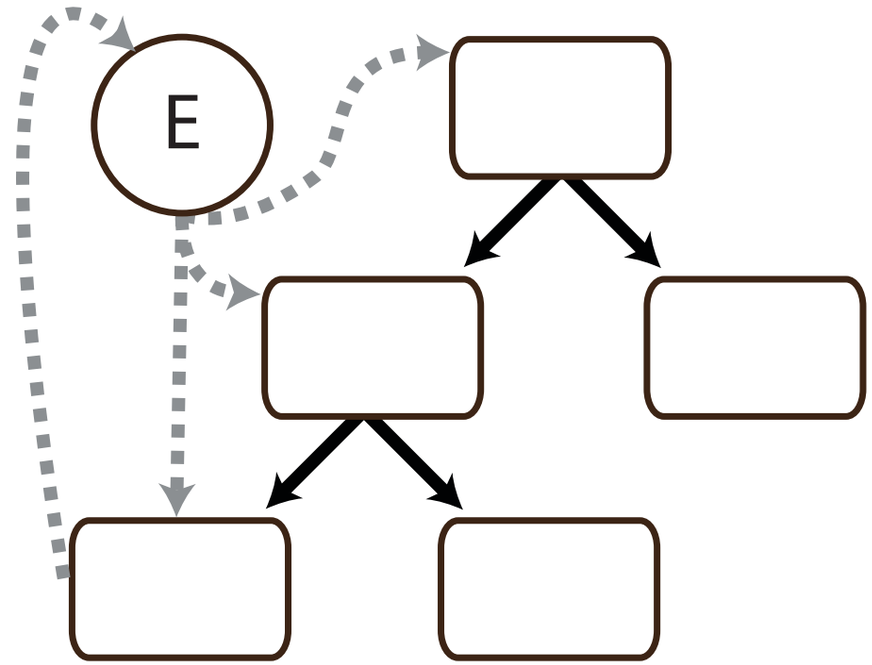
- edge (connection) = data stream

maintains data states and specifications

- executive = coordinate module execution

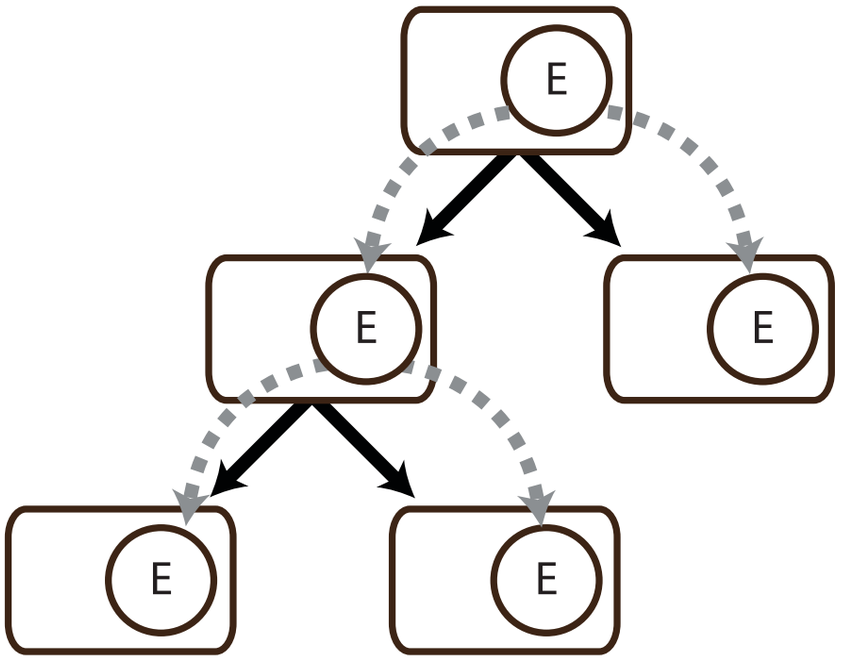


Executive Scope



Centralized

flexible scheduling



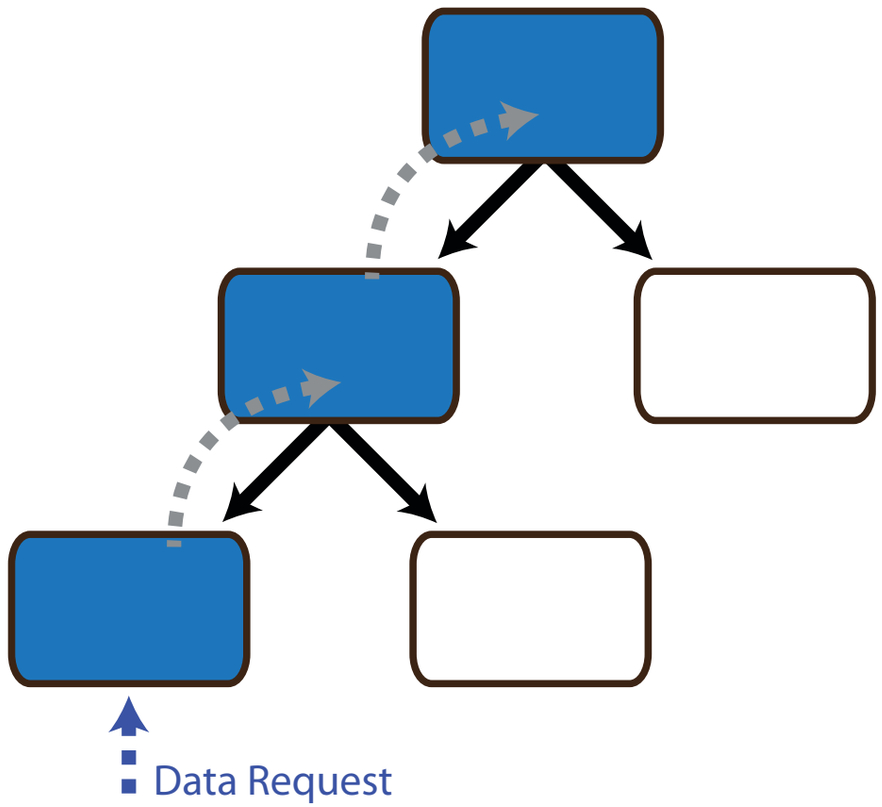
Distributed

good for scalability

Execution Policy

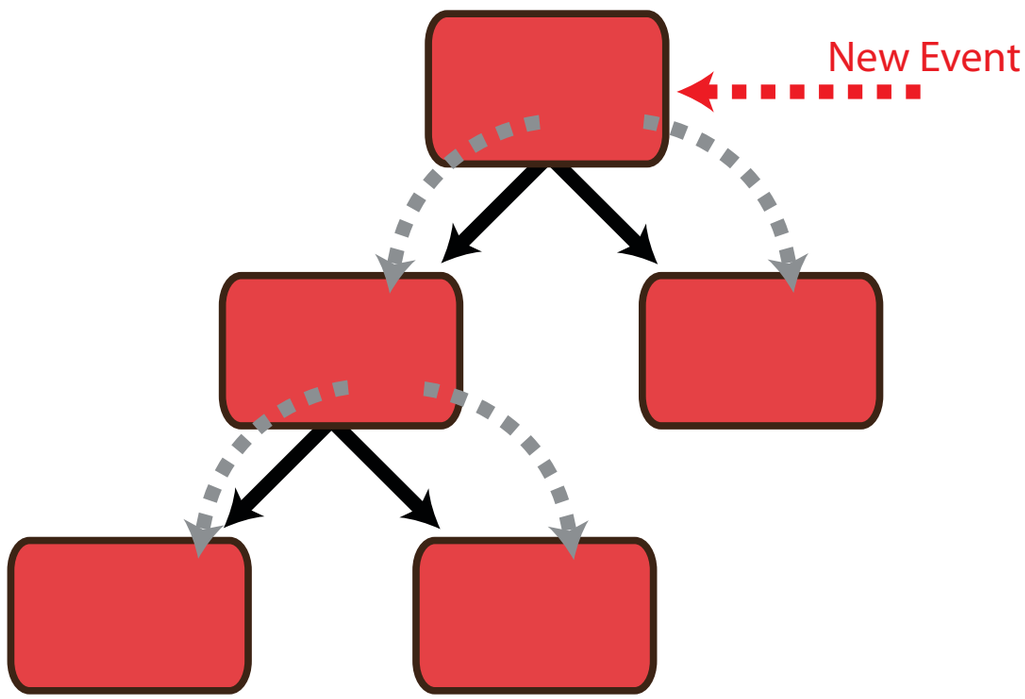
- Policy for communicating between modules

Pull (demand-driven)



minimize computation

Push (event-driven)



minimize coordination

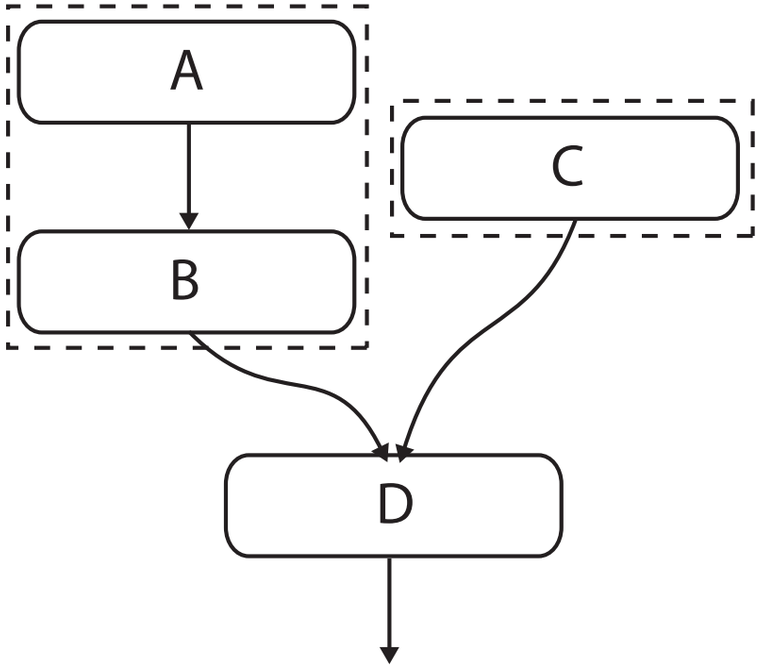
NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

Types of Parallelism

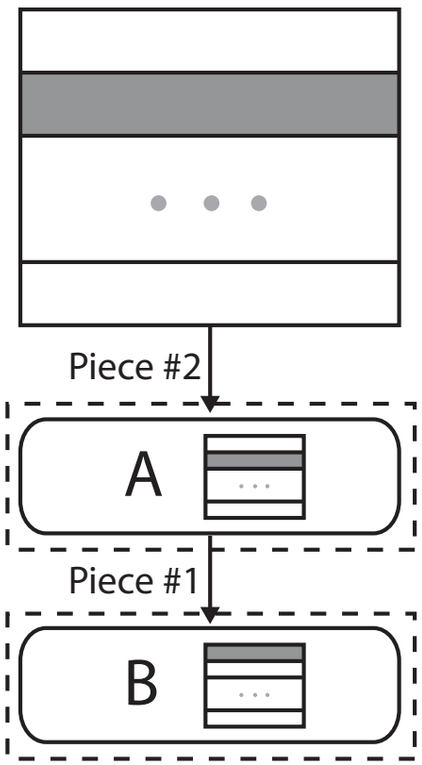
Task

Streamable/Separable Data Structure

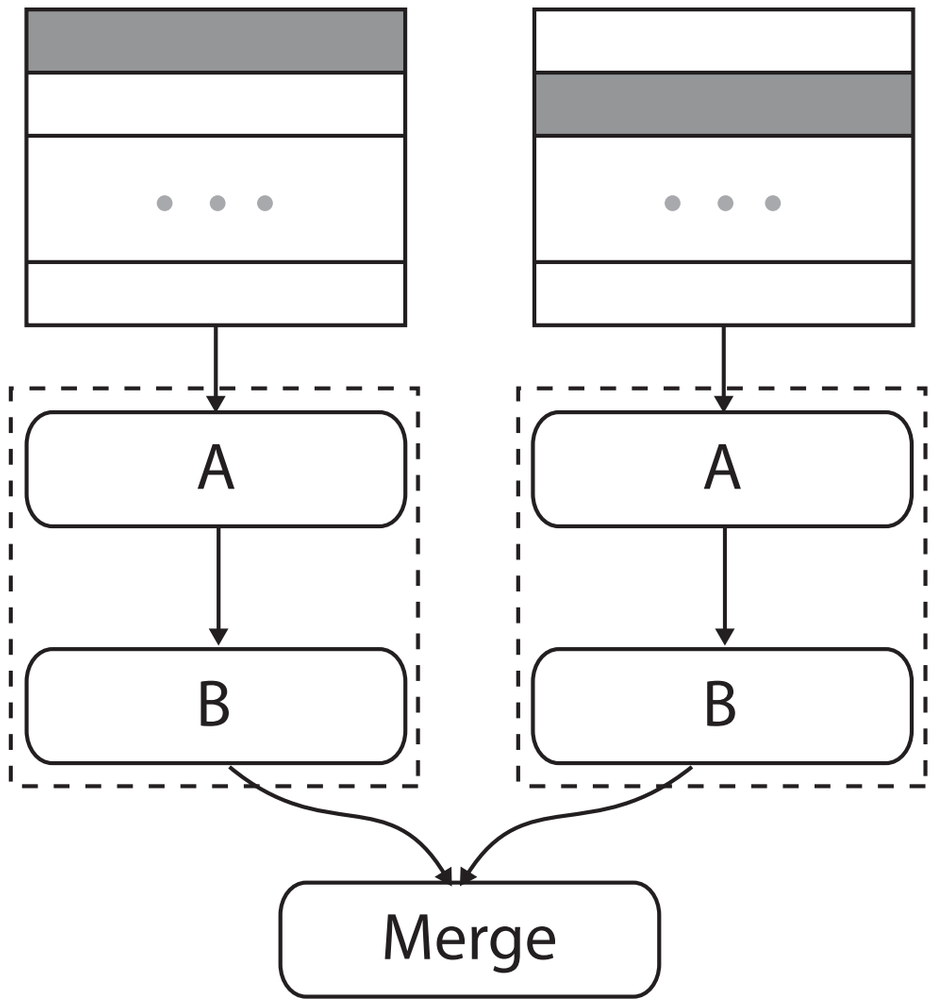
Concurrency Execution



Pipeline



Data



Dataflow Visualization Systems

System	Parallelism	Policy	Scope
VTK		Pull	Dist.
ParaView	Data	Pull	Dist.
Visit	Data + Task	Pull	Dist.
DeVIDE		Pull/Push	Cent.
SCIRun	Task	Push	Cent.
VisTrails		Pull	Cent.

AVS

SCIRun

VTK

VisIt

ParaView
Parallel Visualization Application

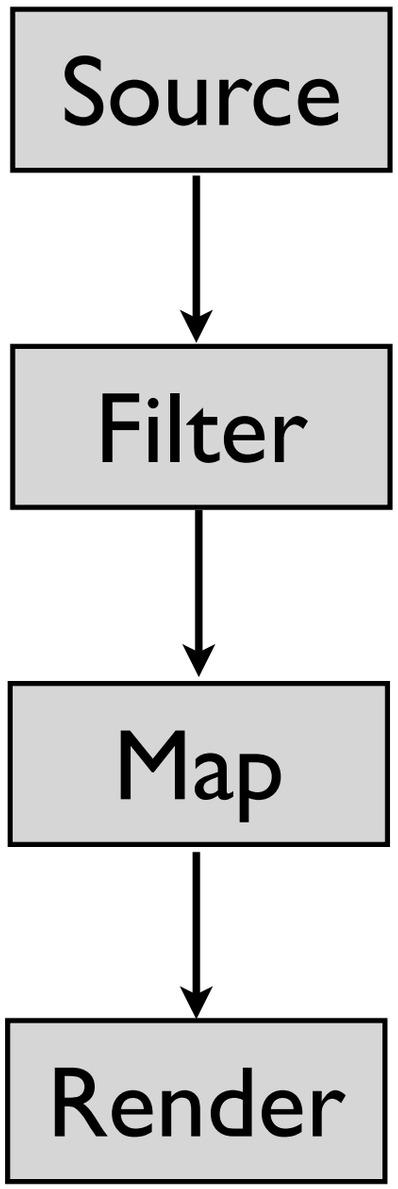
DeVIDE

VisTrails

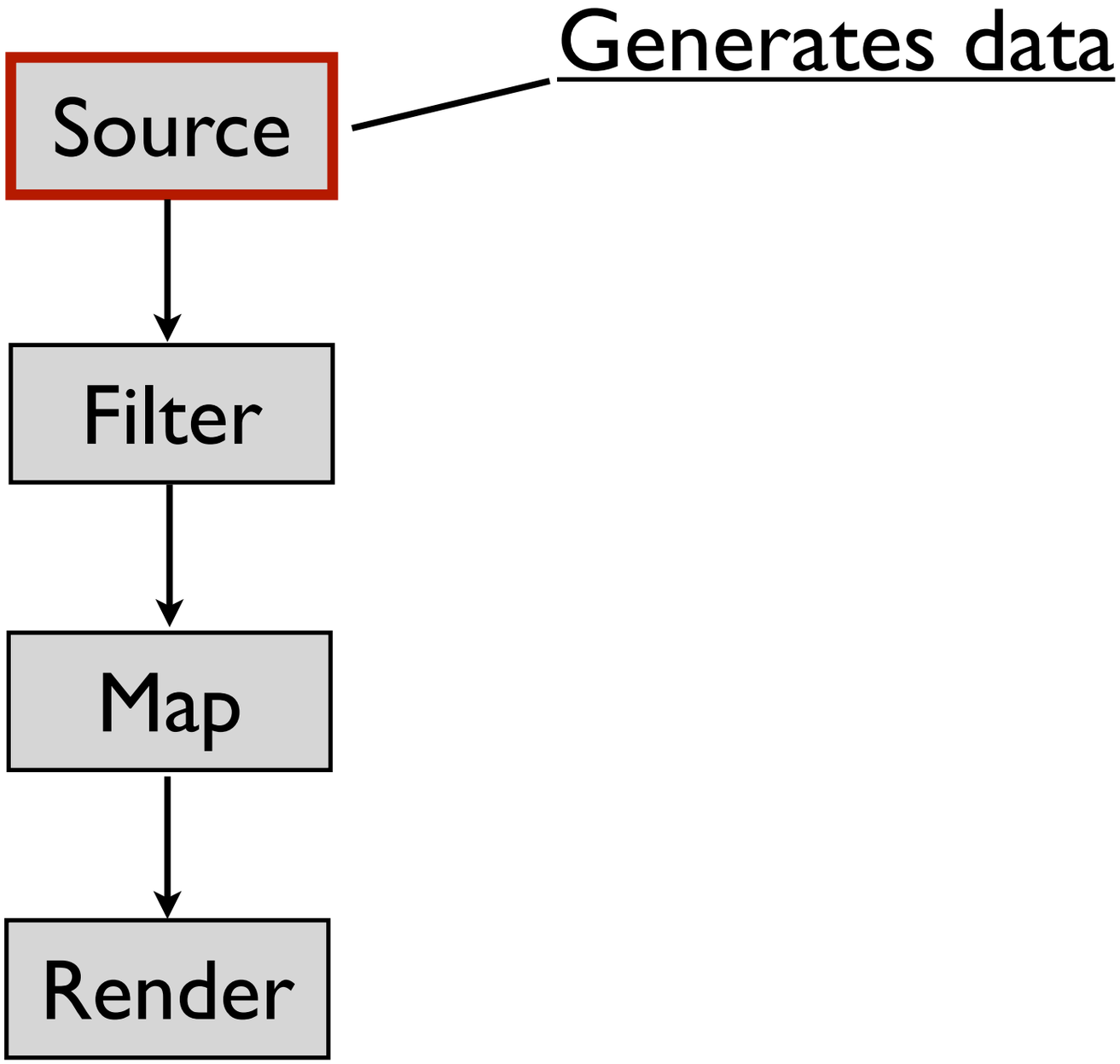
voreen
volume rendering engine

NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

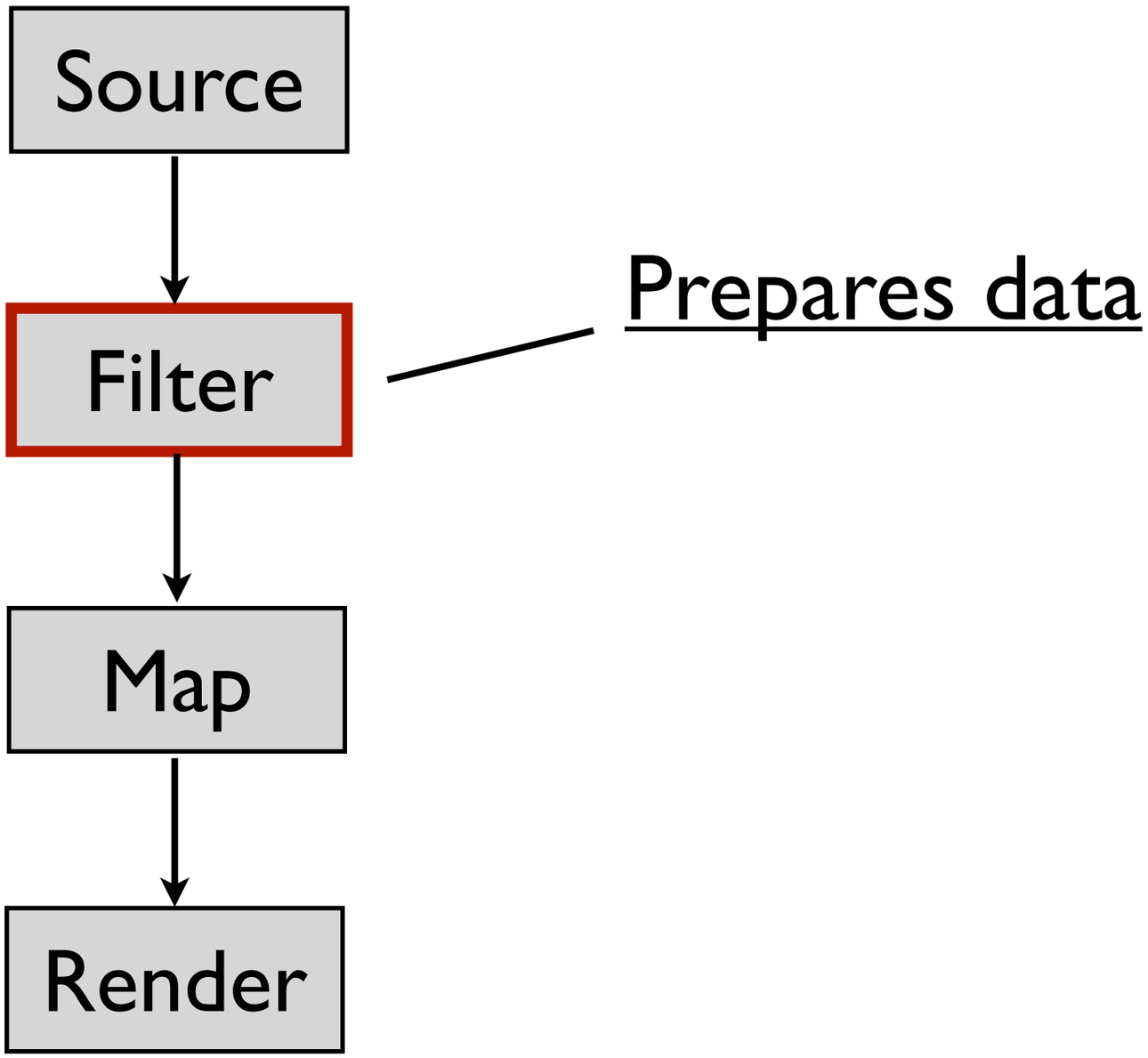
The Visualization Pipeline



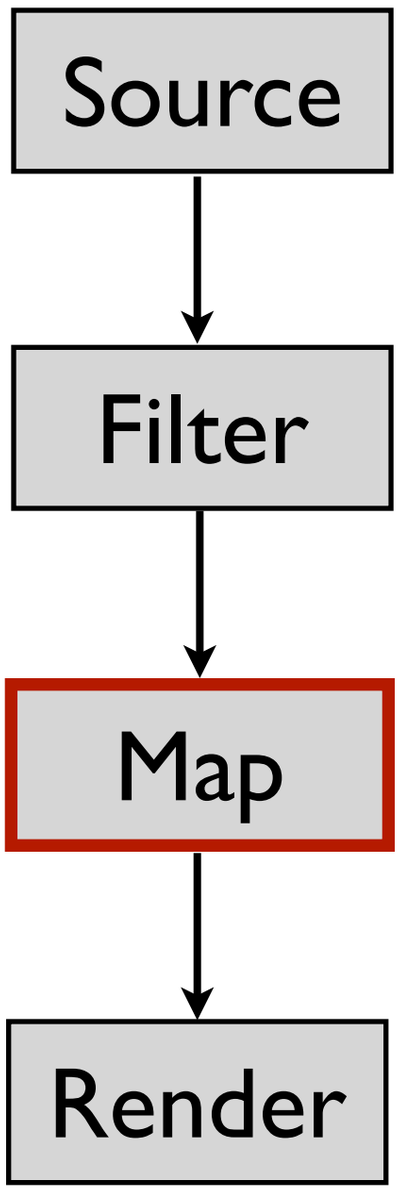
The Visualization Pipeline



The Visualization Pipeline

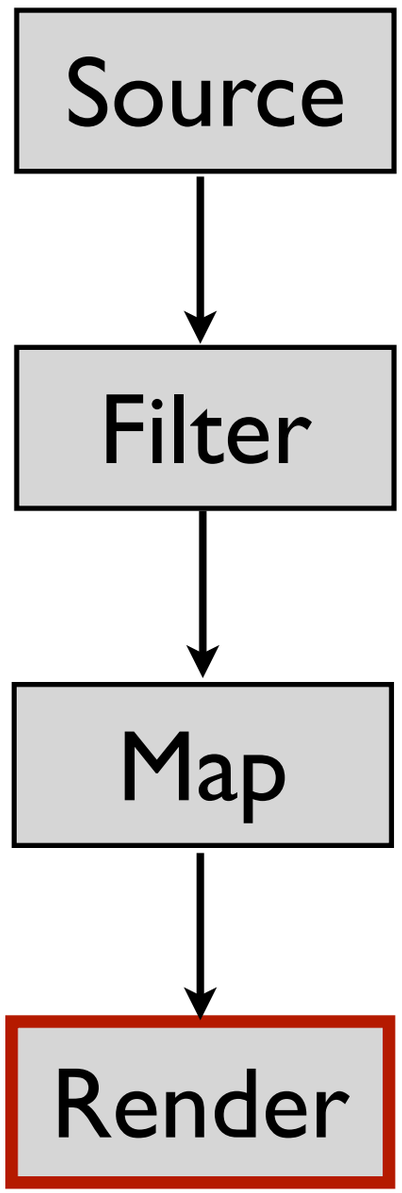


The Visualization Pipeline



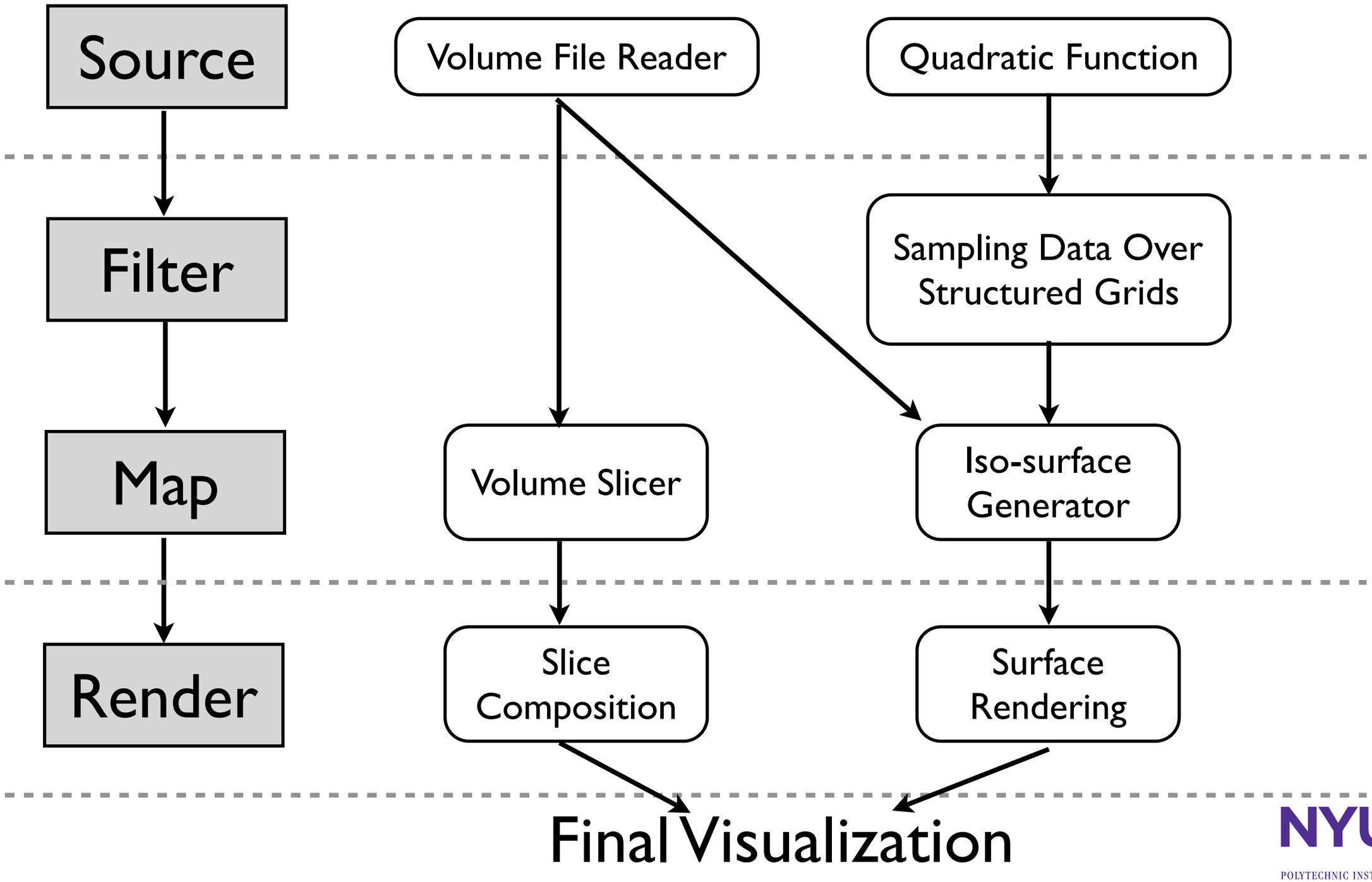
Construct graphics primitives

The Visualization Pipeline



Create actual images

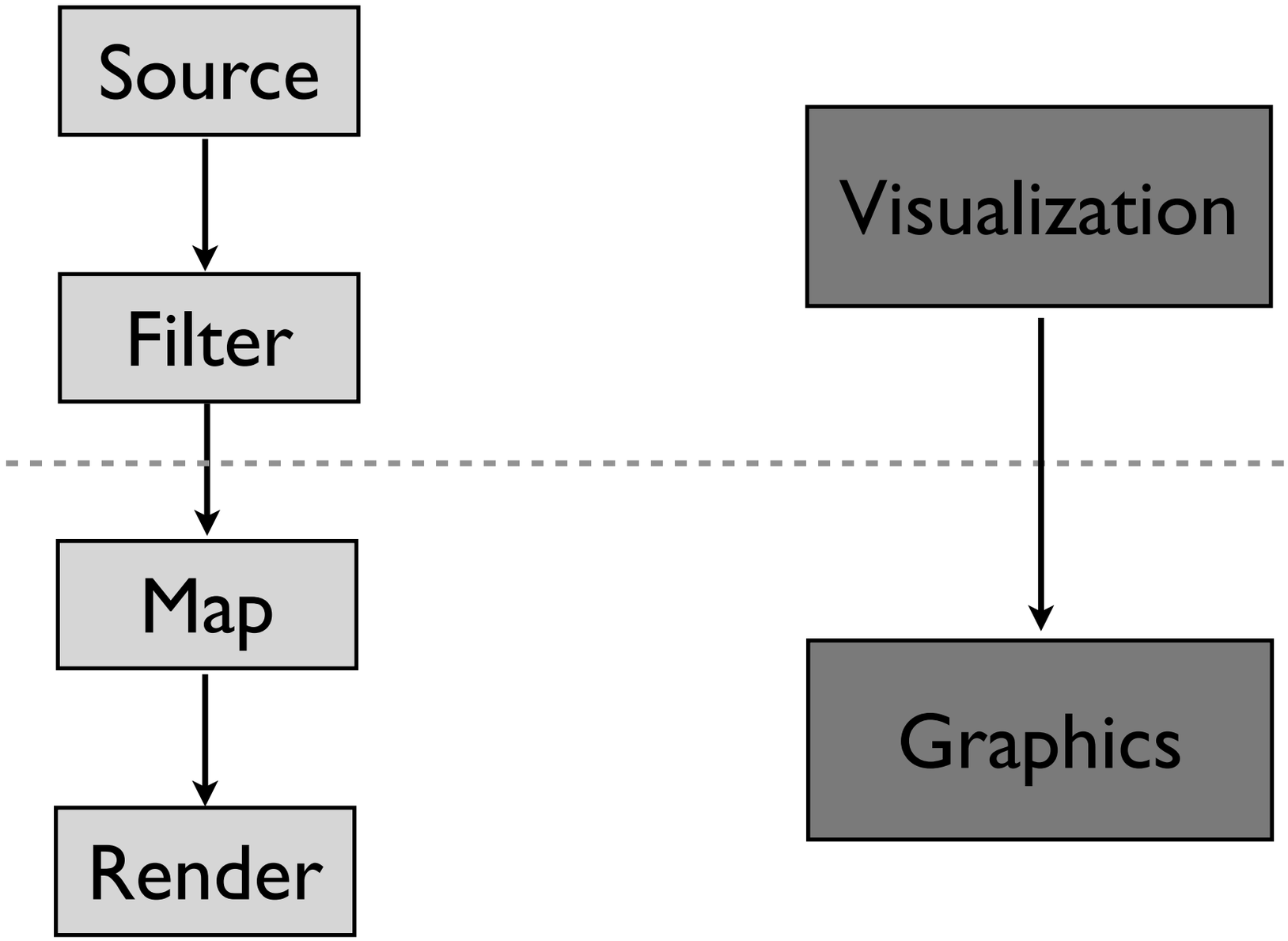
The Visualization Pipeline



The Visualization Toolkit (VTK)

- One of the most popular visualization packages
- Provide pieces to build complex applications
- Open-source, considerably cross-platform
- Written in C++ but has Python/Java/Tcl wrappings
- Advanced visualization applications based on VTK: ParaView, VisIt, 3DSlicer, MayaVi, DeVIDE

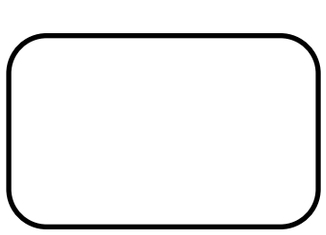
VTK Pipeline



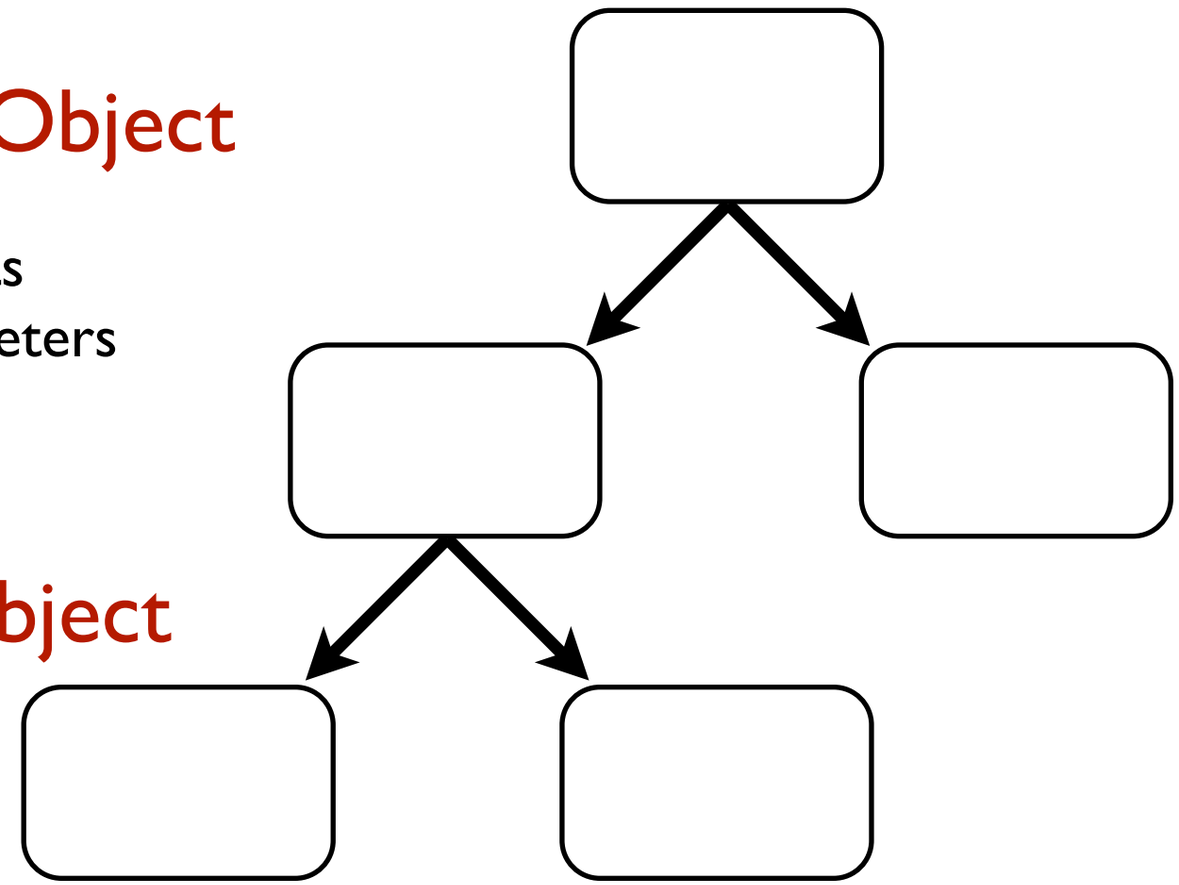
actual
dataflow
network

VTK Visualization Pipeline

- A directed graph
  - node (module) = Algorithm Object



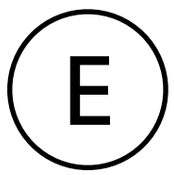
knows how to compute outputs
given a set of inputs and parameters



  - edge (connection) = Data Object

maintains data states and specifications

  - executive = Executive Object



VTK Algorithm Object
`vtkAlgorithm`

- Operates on data objects to produce new data objects
- Base classes for all sources, filters and some intermediate mappers
- Maintain module specifications, e.g. the number of input and output ports
  - Could be zero (for readers and writers)

VTK Data Object
`vtkDataObject`

- General representation of visualization data
  - Holding metadata to support multi-pass execution
- Base classes for all data types

VTK Executive Object
`vtkExecutive`

- Distributed executive
  - Each executive controls exactly one algorithm
- Holding metadata to support demand-driven execution, e.g. update timestamp
- Only knows immediate executives that it connects to

Using `vtkAlgorithm` vs. `vtkExecutive`

- Can be used interchangeably through the API since they have one-to-one mapping
  - Underneath VTK directs the correct calls to either vtkAlgorithm or vtkExecutive
- Chain SetInputConnection() and GetOutputPort() calls to construct VTK dataflow networks

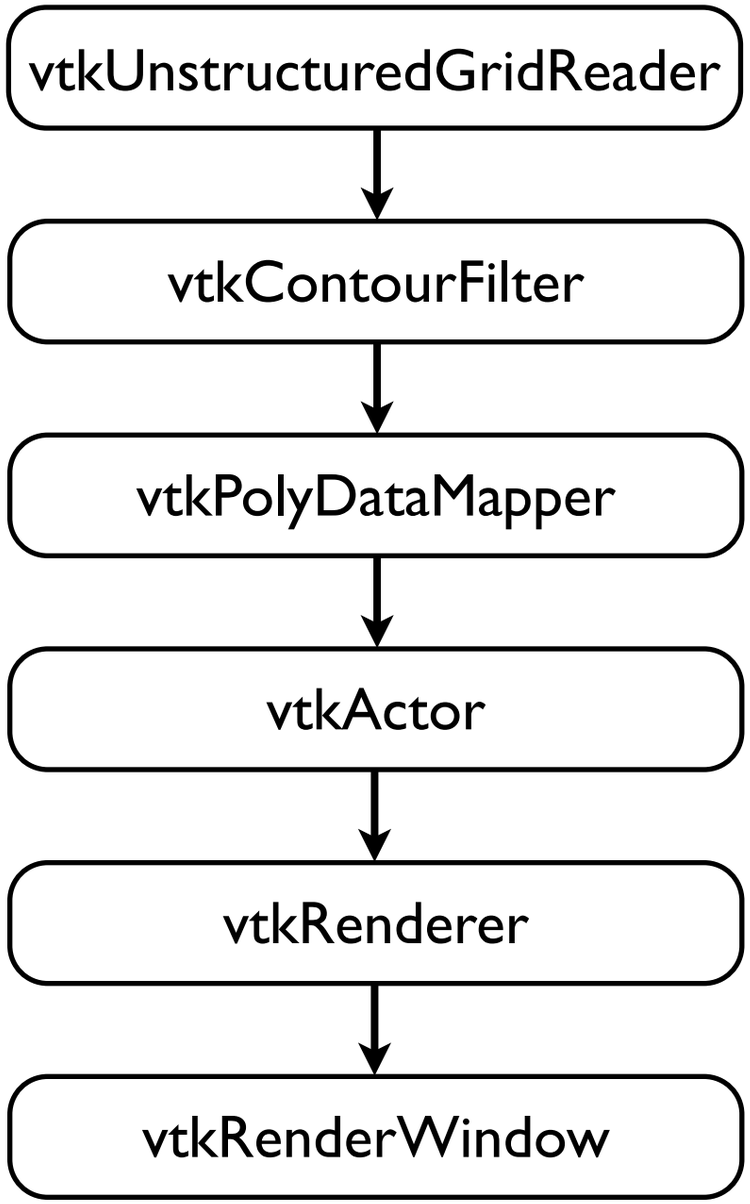
VTK Graphics Pipeline

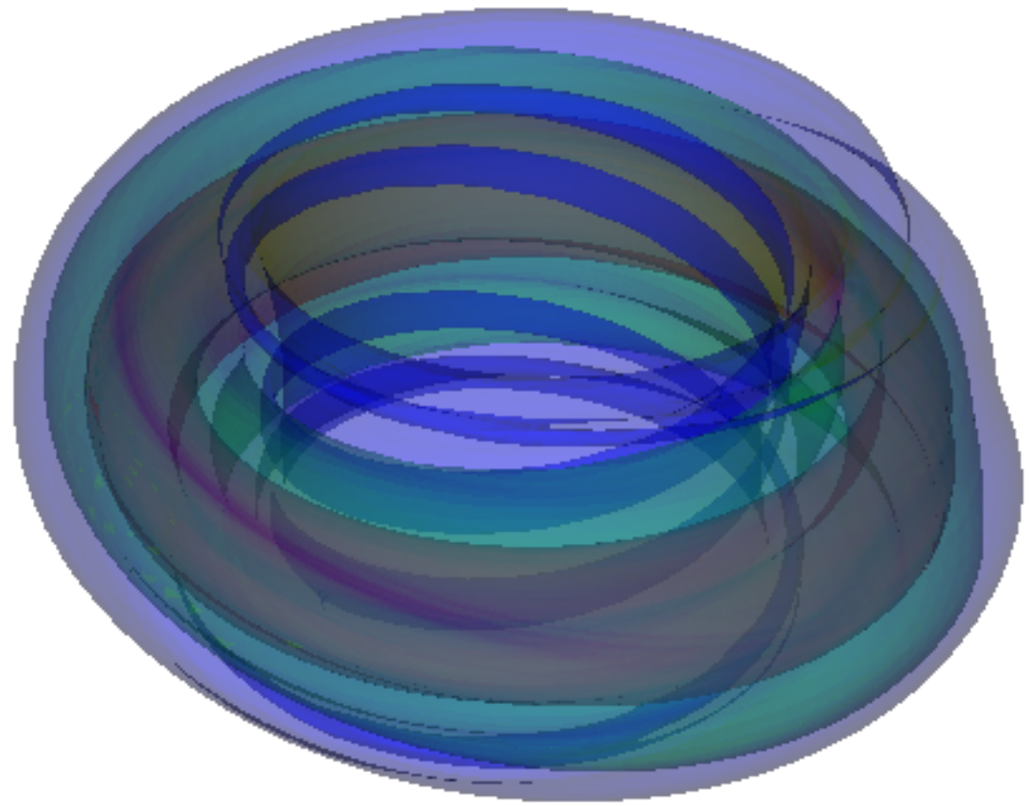
- Mapper: produce geometries, interfacing between the visualization and graphics phase of VTK
- Properties: rendering properties, e.g. color, material, etc.
- Actor: scene objects (geometry+properties)
- Renderer: specify the rendering logic with lights, camera, etc.
- Render Window and Interactor: manages windows and user interactions

VTK Pipeline Execution

- Pull/Demand-Driven
- Triggered by
  - mappers from the graphics phase requested geometry from the visualization dataflow
  - explicitly called to Update()
- Each update will propagate up to sources
- Not thread-safe

Example


vtkUnstructuredGridReader
vtkContourFilter
vtkPolyDataMapper
vtkActor
vtkRenderer
vtkRenderWindow

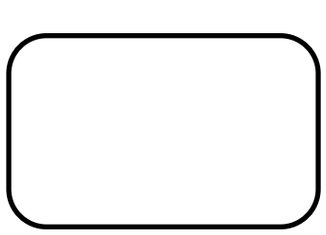


VisTrails

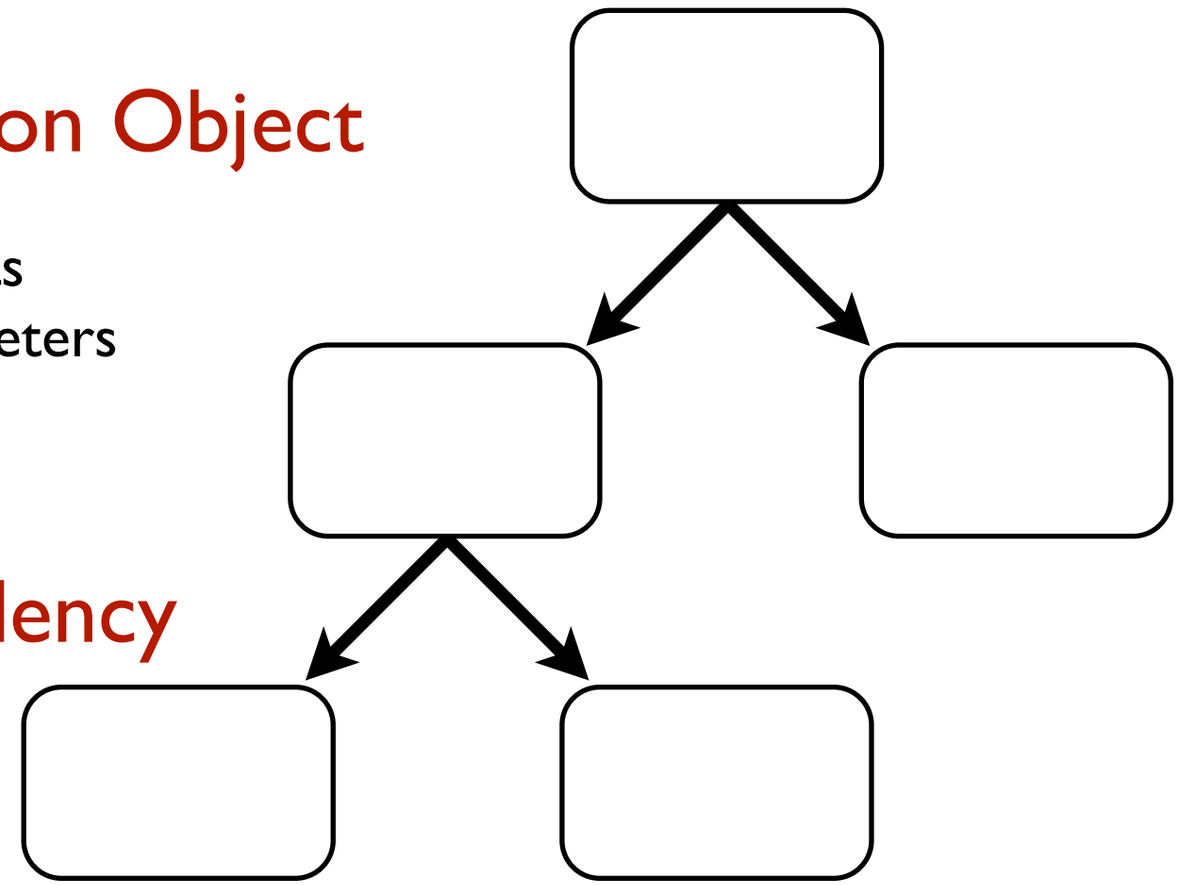
- Visual programming interface
  - Simplify the pipeline creation process
- Capture detailed provenance
- Provide comparative visualization through the spreadsheet
- Python-based
  - Seamlessly integrate with many libraries

VisTrails Visualization Pipeline

- A directed graph
  - node (module) = Computation Object



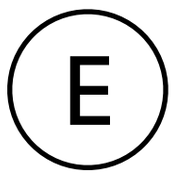
knows how to compute outputs
given a set of inputs and parameters



  - edge (connection) = Dependency

maintains data states and specifications

  - executive = Interpreter



NYU:poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

VisTrails Computation Object

- A python class with a compute() method
  - VTK is wrapped by calling Update() method inside compute()
- Data stays at the source module (the module that generates it)
- Maintain metadata such as the number of input/ output ports and annotations

VisTrails Connection Object

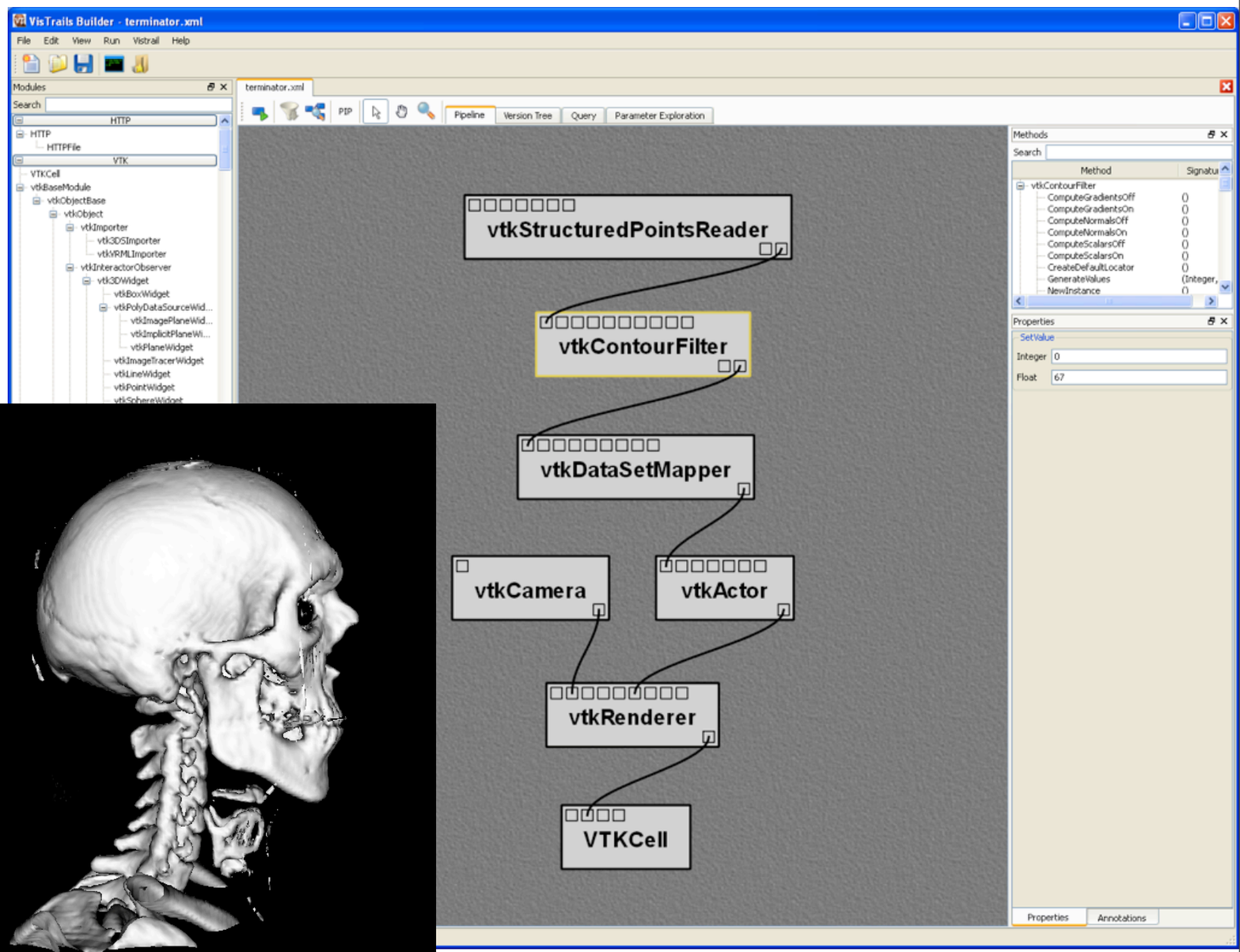
- Does not hold actual data
- Only keeps the information on the two ports that it connects to ensure correct execution order

VisTrails Interpreter

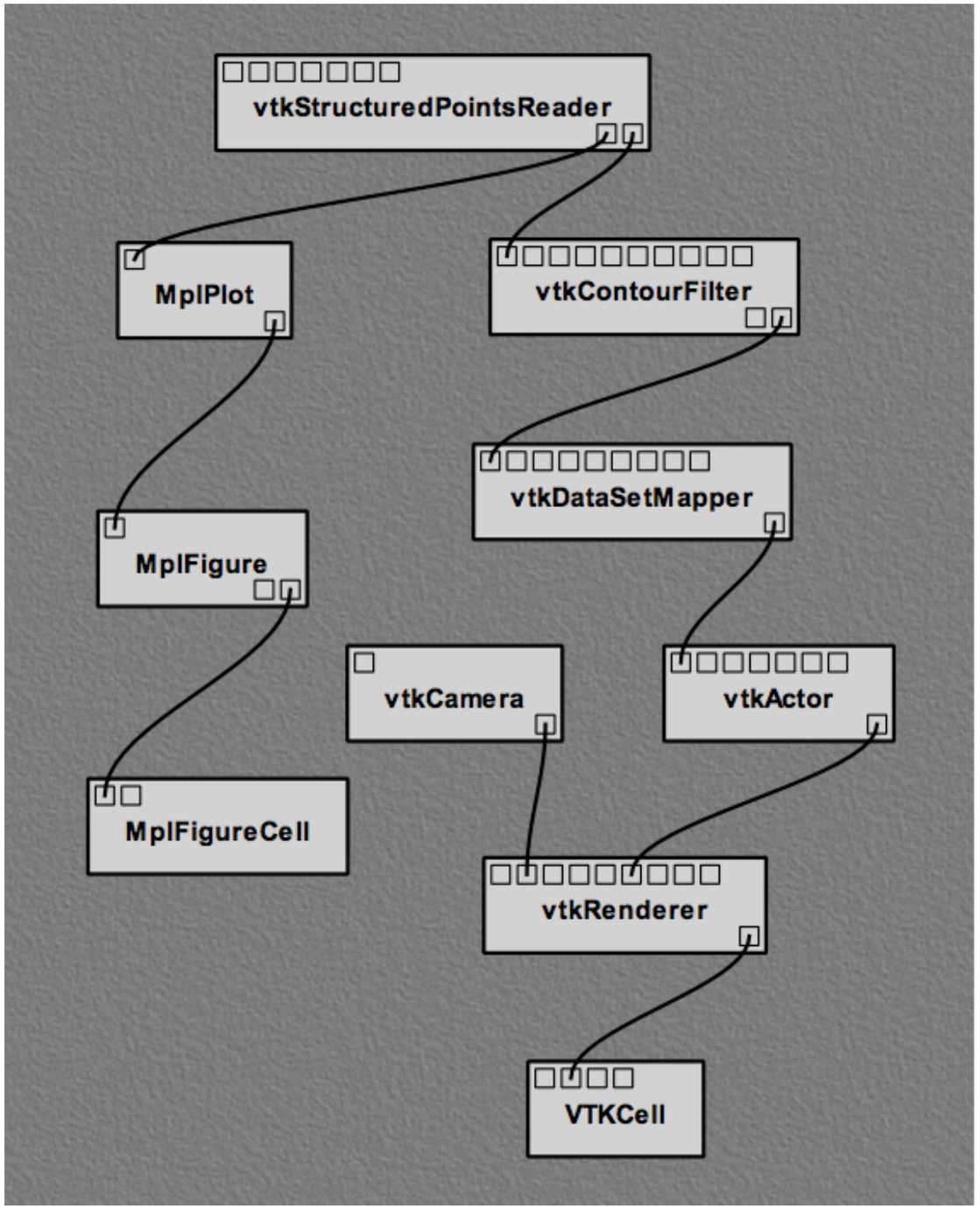
- Centralized executive
- Ensure the whole pipeline execution from sinks to sources (a BFS from sinks)
- Maintain output caches to speed up computation

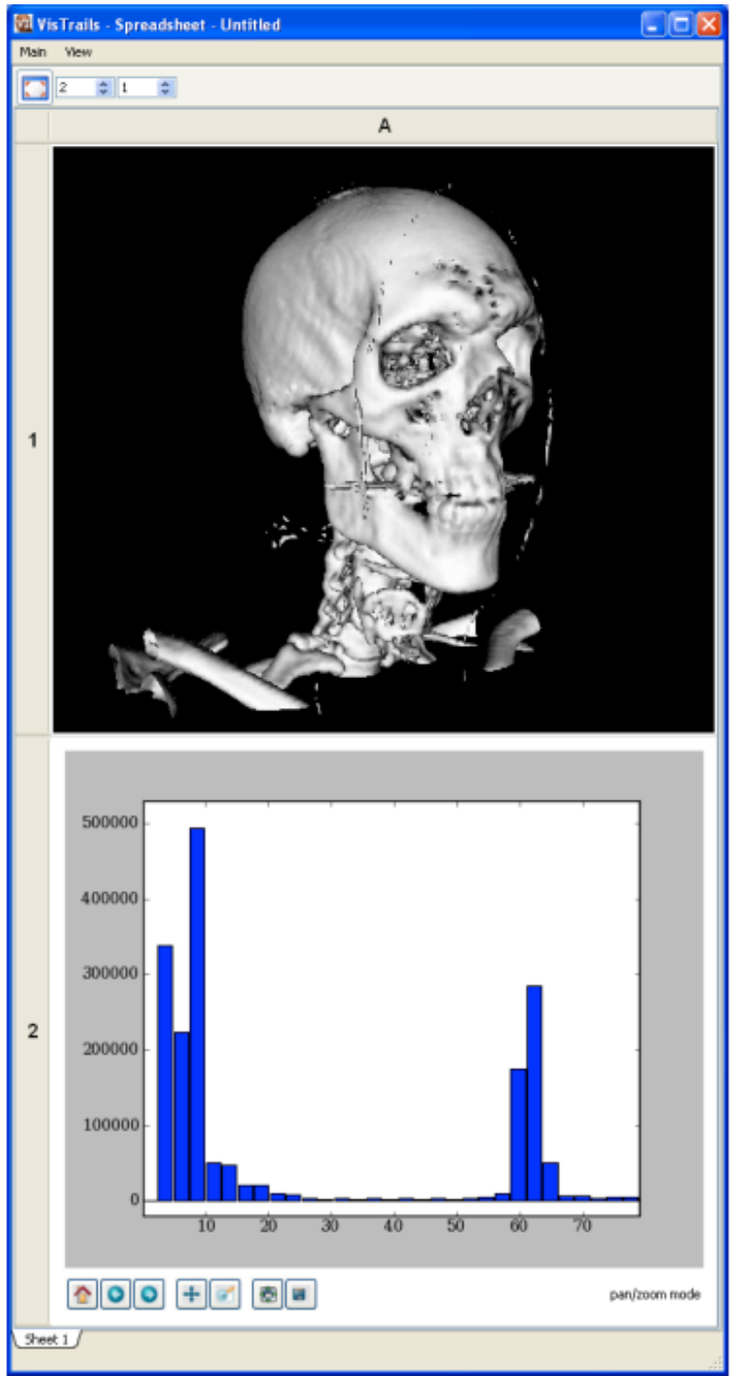
Example

```
import vtk
data = vtk.vtkStructuredPointsReader()
data.SetFileName("..examples/data/head.120.vtk")

contour = vtk.vtkContourFilter()
contour.SetInput(0,data.GetOutput())
contour.SetValue(0, 67)

mapper = vtk.vtkPolyDataMapper()
mapper.SetInput(contour.GetOutput())
mapper.ScalarVisibilityOff()

actor = vtk.vtkActor()
actor.SetMapper(mapper)

ren = vtk.vtkRenderer()
ren.AddActor(actor)
renwin = vtk.vtkRenderWindow()
renwin.AddRenderer(ren)

style = vtk.vtkInteractorStyleTrackballCamera()
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renwin)
iren.SetInteractorStyle(style)
iren.Initialize()
iren.Start()
```

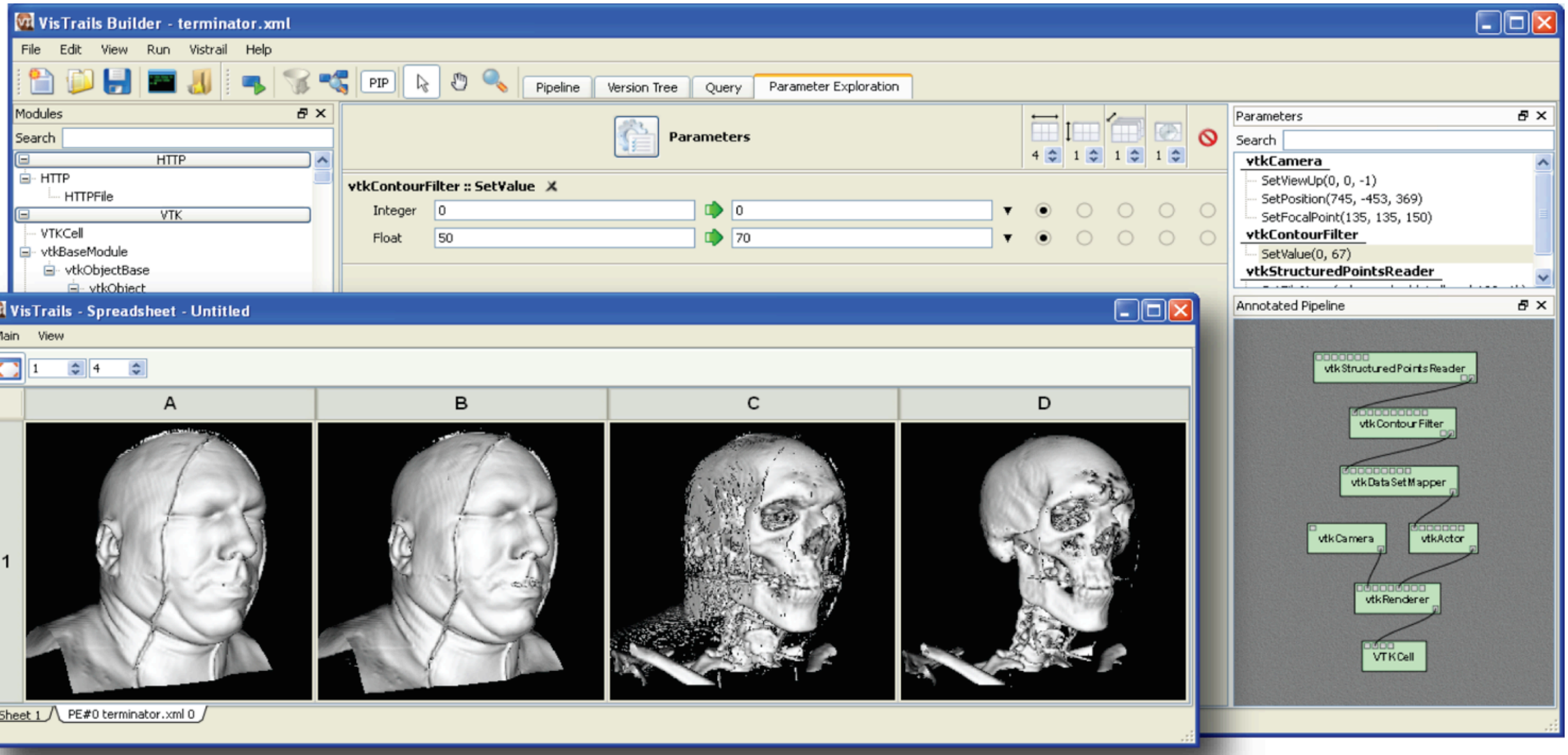


Example


vtkStructuredPointsReader
MplPlot
vtkContourFilter
vtkDataSetMapper
MplFigure
vtkCamera
vtkActor
MplFigureCell
vtkRenderer
VTKCell


VisTrails - Spreadsheet - Untitled
Main
View
A
1
2
500000
400000
300000
200000
100000
0
10
20
30
40
50
60
70
pan/zoom mode
Sheet 1

Example



NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

Quick Guide on Visualization Tools

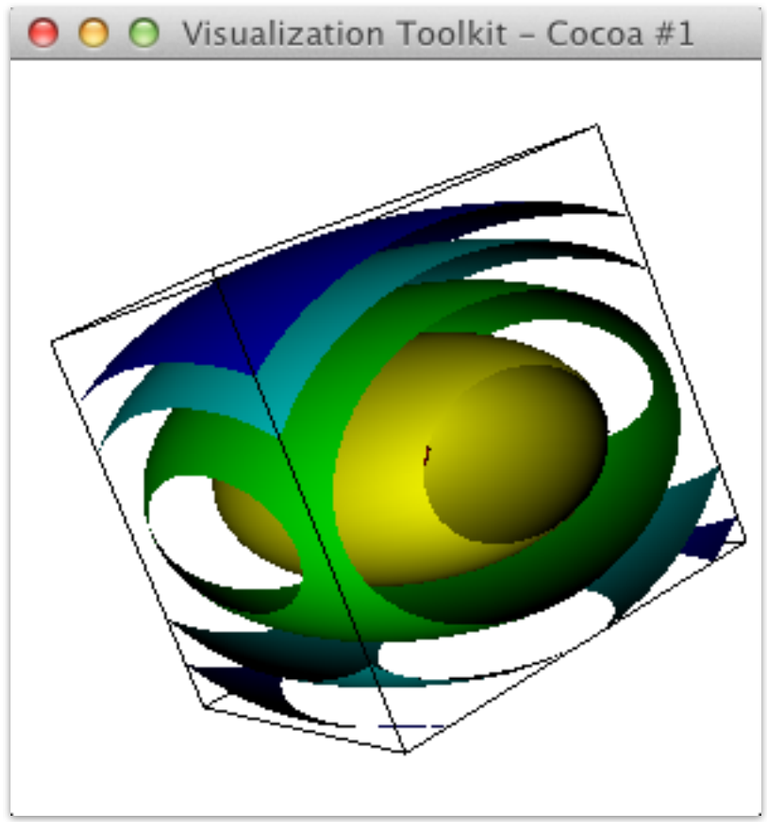
- VTK with C++ and CMake
- Python VTK, scripting programming
- VisTrails, visual programming

Build VTK from Source

- Required for Python/Java wrapping and for Linux/Mac distribution
- Recommended the trunk for development
- Cross-platform compilation using CMake

Exercise 1

- Compile VTK with Python support
- Testing: run Examples/VisualizationAlgorithms/Python/VisQuad.py



CMake

- Open-source, cross-platform Make
  - Out-of-source build
- Natively support linking with VTK, i.e. find_package(VTK)
- Tutorial available at www.cmake.org

Exercise 2

- Build the same program in Exercise 1 but using C++ and CMake

VisTrails

- Binaries and user guides are available at www.vistrails.org

NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

Exercise 3

- Build the same pipeline in VisTrails
- Perform parameter exploration

NYU·poly
POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY