Systems Architecture for Visualisation

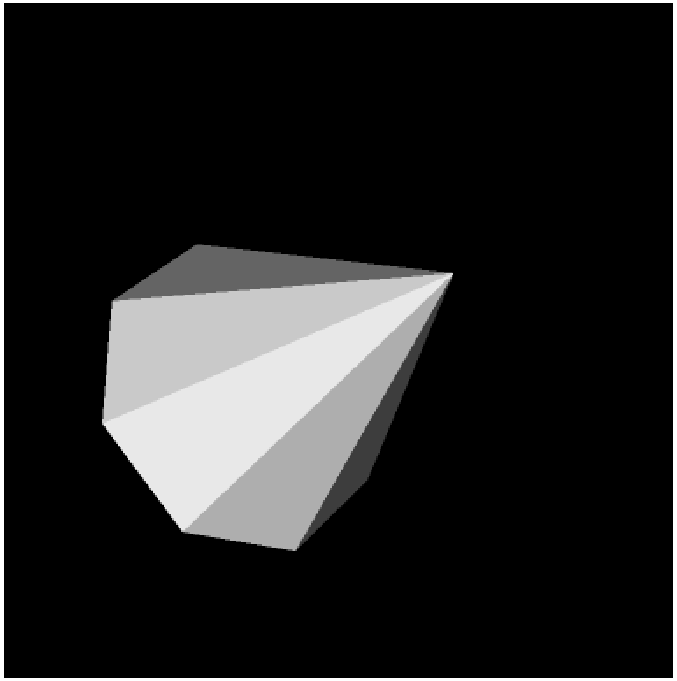
Visualisation – Lecture 4

Taku Komura

Institute for Perception, Action & Behaviour
School of Informatics

Last lecture ....

- Basics of **Computer Graphics**
  - as we need them for **visualisation**
    - **object representation**
    - **object illumination**

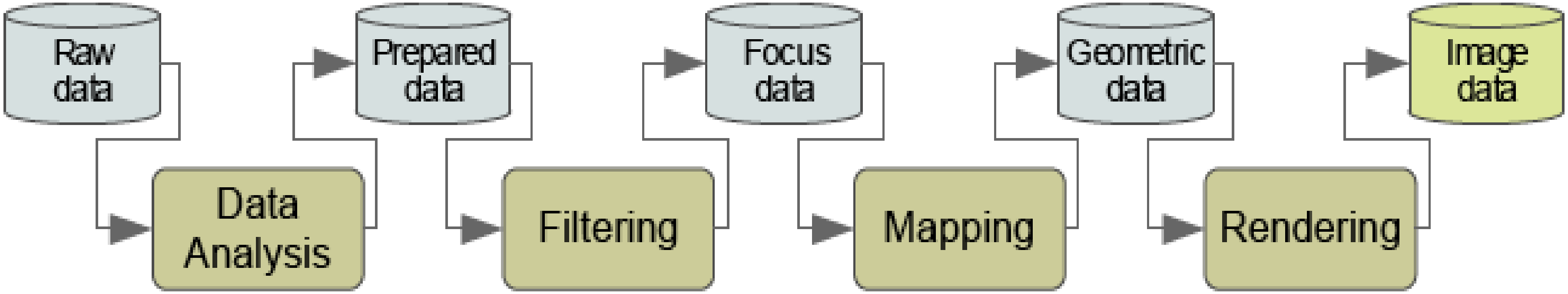


VTK cone example

This lecture

- **The Visualisation Pipeline**
  - systems architecture for visualisation
    - outline
    - pipeline connections
    - pipeline execution

Visualisation Pipeline - Overview

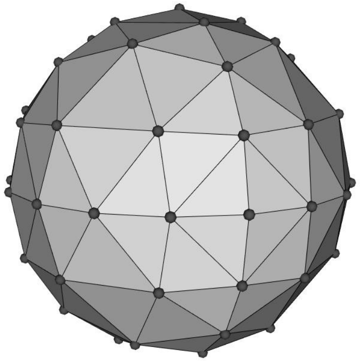


- Three main elements
  - objects to represent data (**data objects**)
  - objects to represent processes (**process objects**)
  - **direction of data flow**
    - indicates data dependencies
    - synchronisation required to keep pipeline up to date

Visualisation Pipeline – Objects 1

- Data Objects
  - **represent data** (internally) + methods to access it
  - **data modification** only via formal object methods
  - additional **data properties for rendering**

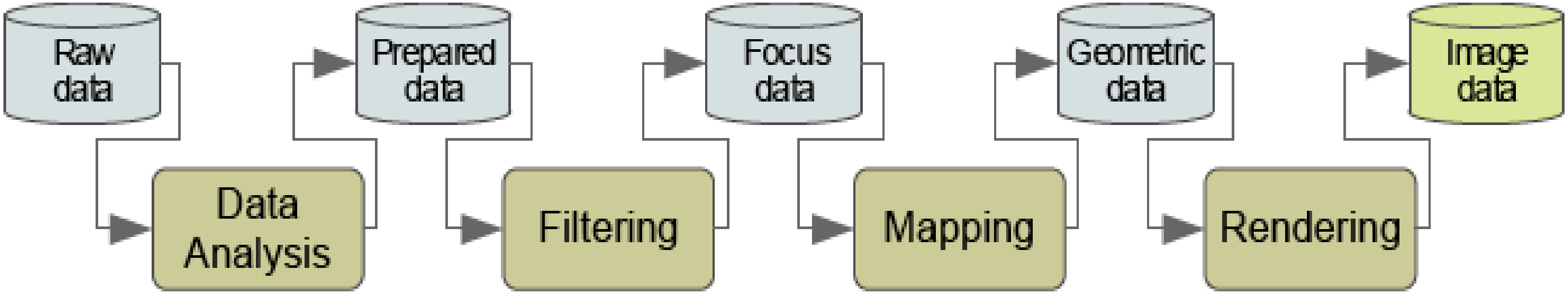
  - *Example* : mesh
    - vertices, connectivity (basic)
    - polygons, normals at vertices or faces (additional)



Visualisation Pipeline – Objects 2

- Process Objects
  - objects that operate on input data to generate output data
  - **data transformation**
  - **source objects**
    - **generate data** from local parameters (e.g. quadric) or external source (e.g. file)
  - **Data analyser**
  - **Filter objects**
  - **mapper objects**
    - transform data into graphic primitives (for display or file output)

Data Analyser

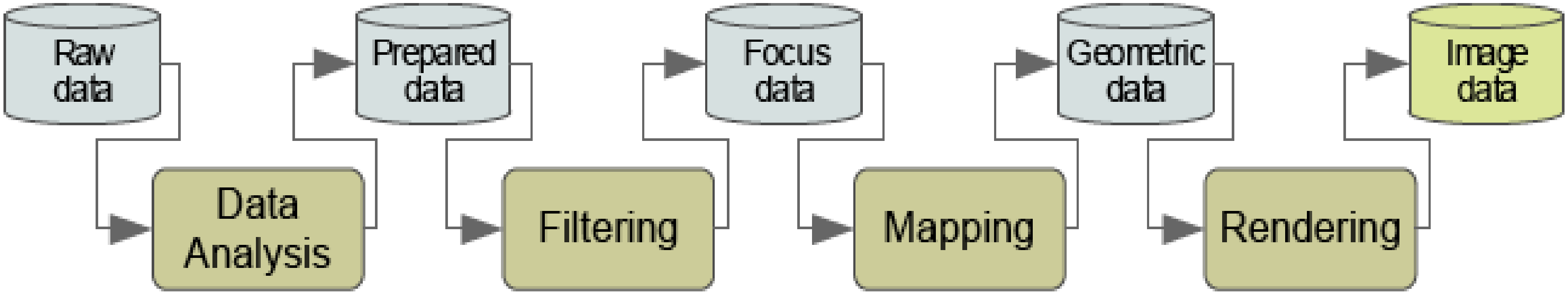


Data Analysis: data are prepared for visualization

- applying a smoothing filter,
- interpolating missing values,
- or correcting erroneous measurements

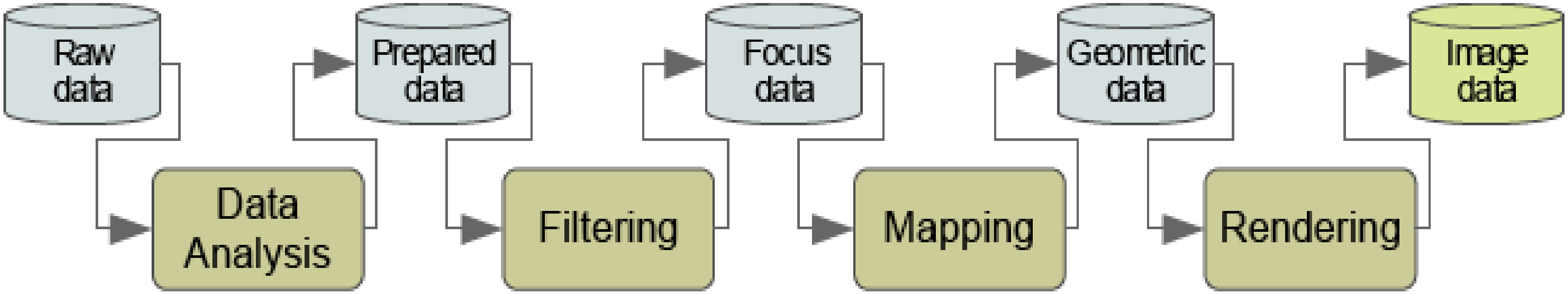
usually computer-centered, little or no user interaction.

Data Filtering



Filter Objects: selection of data portions to be visualized -- usually user-centered.

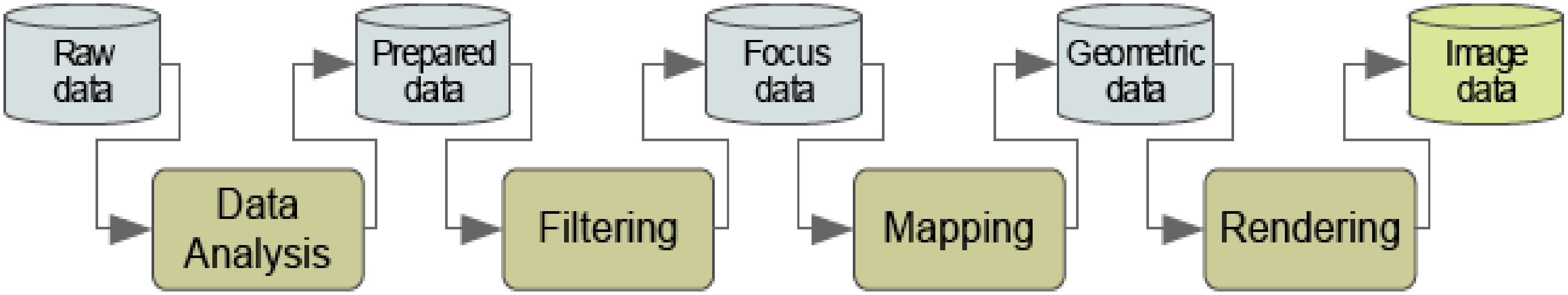
Mapping and Rendering



Mapper Objects : data are mapped to geometric primitives (e.g., points, lines) and their attributes (e.g., color, position, size);

Rendering: geometric data are transformed to image data.

Visualisation Pipeline - Overview



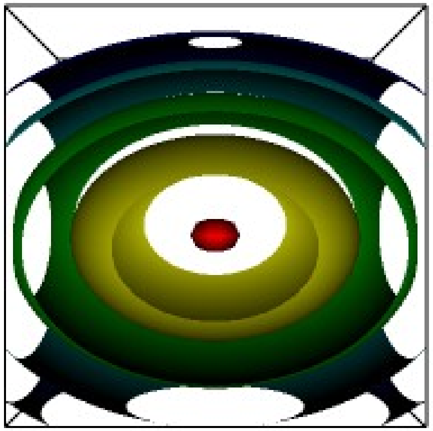
- The modules do not have to necessarily follow this order
- Especially, the Data analysis and Filtering can appear several times

An example : visualising a quadric

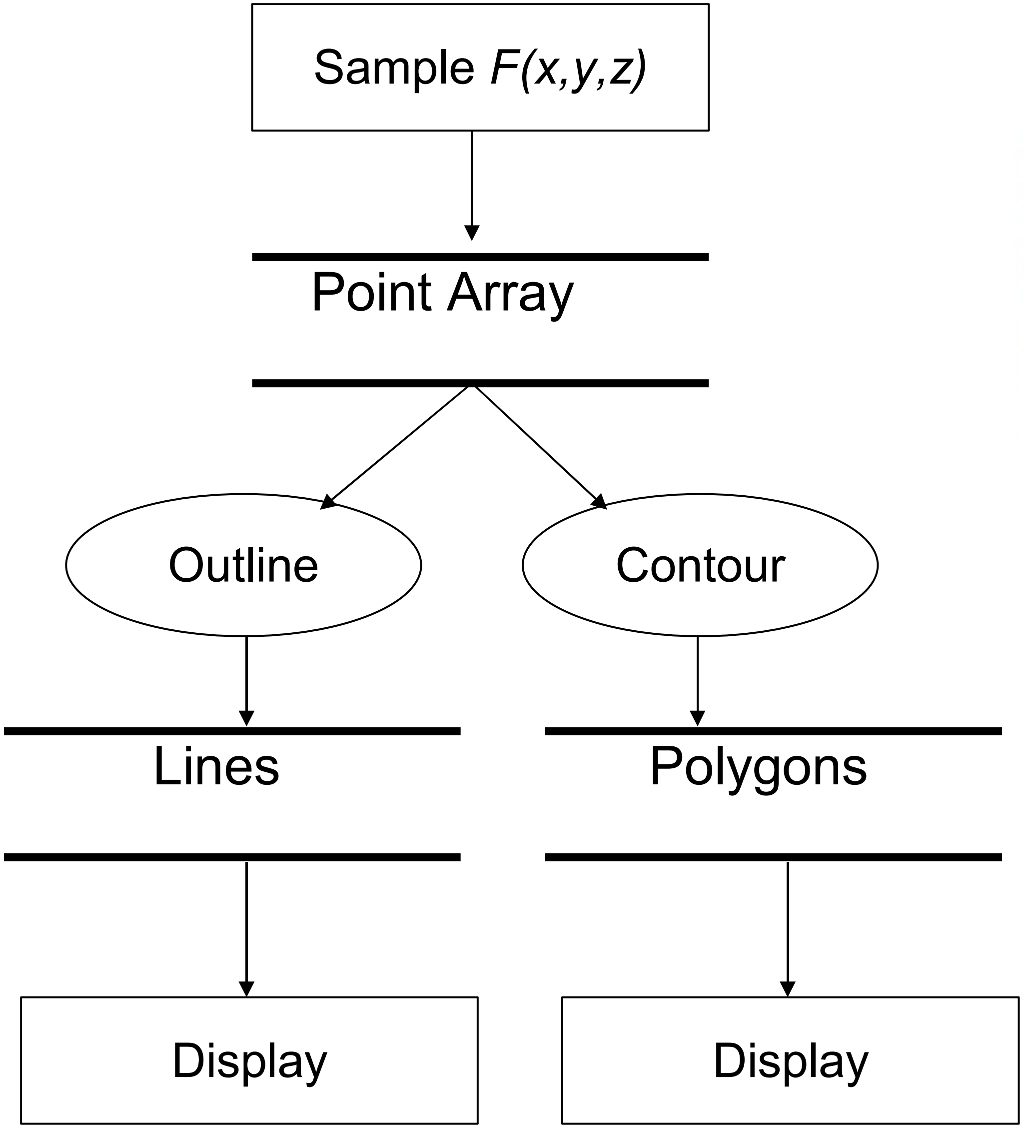
- Quadric
  - second order surface function in $\mathbb{R}^3$ (more than 2 variables in def.)

$$F(x,y,z) = (ax + by + cz + d)(ex + fy + gz + h)$$
$$= a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9$$

  - co-efficients : *a,b,c,d,e,f,g,h* variables: *x,y,z*
- **Task:** Visualise a quadric in the region $-1 \leq x, y, z \leq 1$
- **Process :**
  - Evaluate equation on a 30 x 30 x 30 regular grid
  - Extract 5 surfaces corresponding to values of the function *F(x,y,z) = c*.
  - Generate a 3D outline round the data (bounding box)



Visualising a Quadric : Functional Model



Sample *F(x,y,z)*

Point Array

Outline

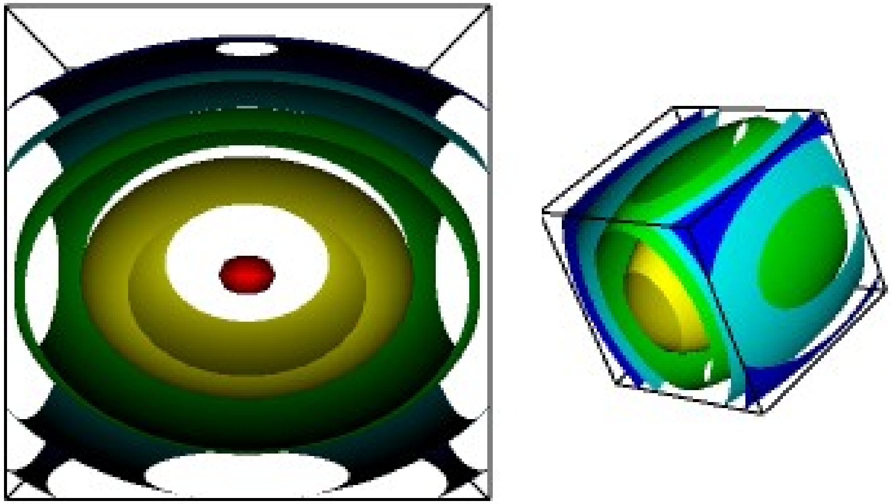
Contour

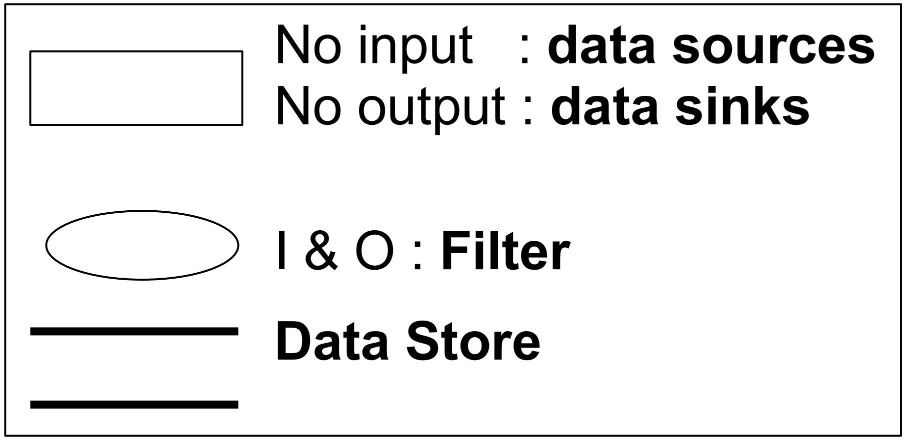
Lines

Polygons

Display

Display





No input : **data sources**

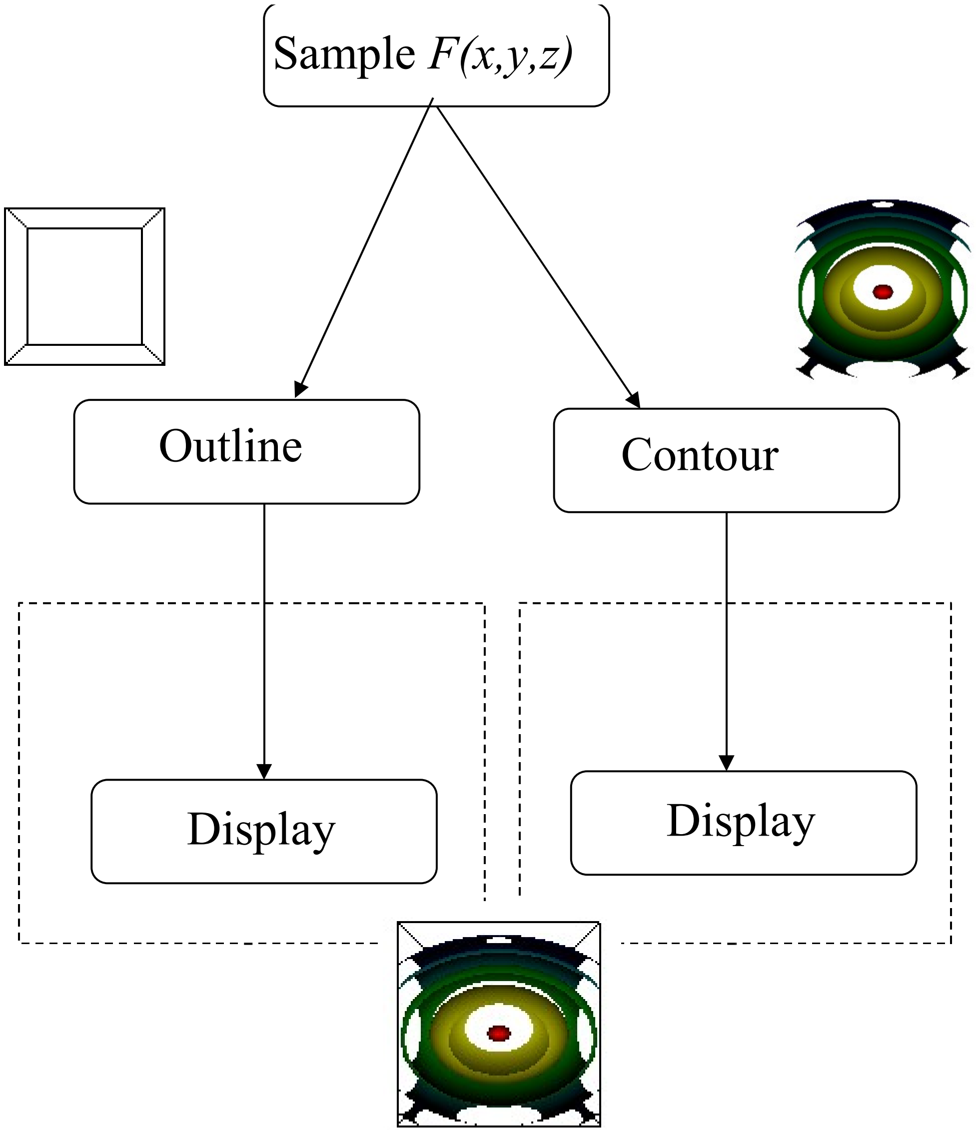
No output : **data sinks**

I & O : **Filter**

Data Store

Visualising a Quadric : process objects

$$F(x,y,z)=(ax+by+cz+d)(ex+fy+gz+h)$$
$$=a_0x^2+a_1y^2+a_2z^2+a_3xy+a_4yz+a_5xz+a_6x+a_7y+a_8z+a_9$$



- Source object
  - procedural generation of quadric
  - vtkQuadric
- Filter Objects
  - `vtkContourFilter` `vtkOutlineFilter`
  - although graphics representation still an internal representation
- Mapper objects
  - conversion to graphics primitives
  - `vtkPolyDataMapper`

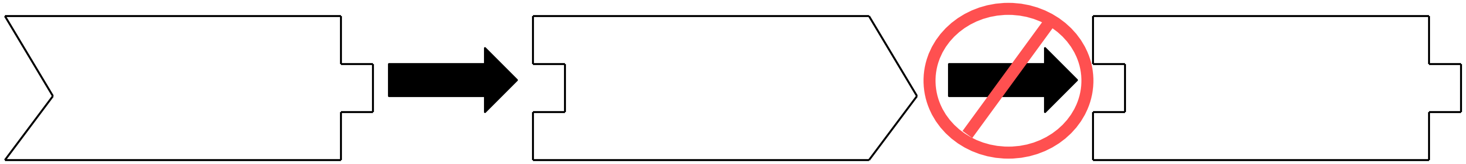
See `VisQuad.tcl`

Visualisation Pipeline Connections

- {Sources, filters, mappers} modules can be connected in variety of ways

- Connectivity Issues:

Type: restrictions on types of data that a module can handle (as input / for output)

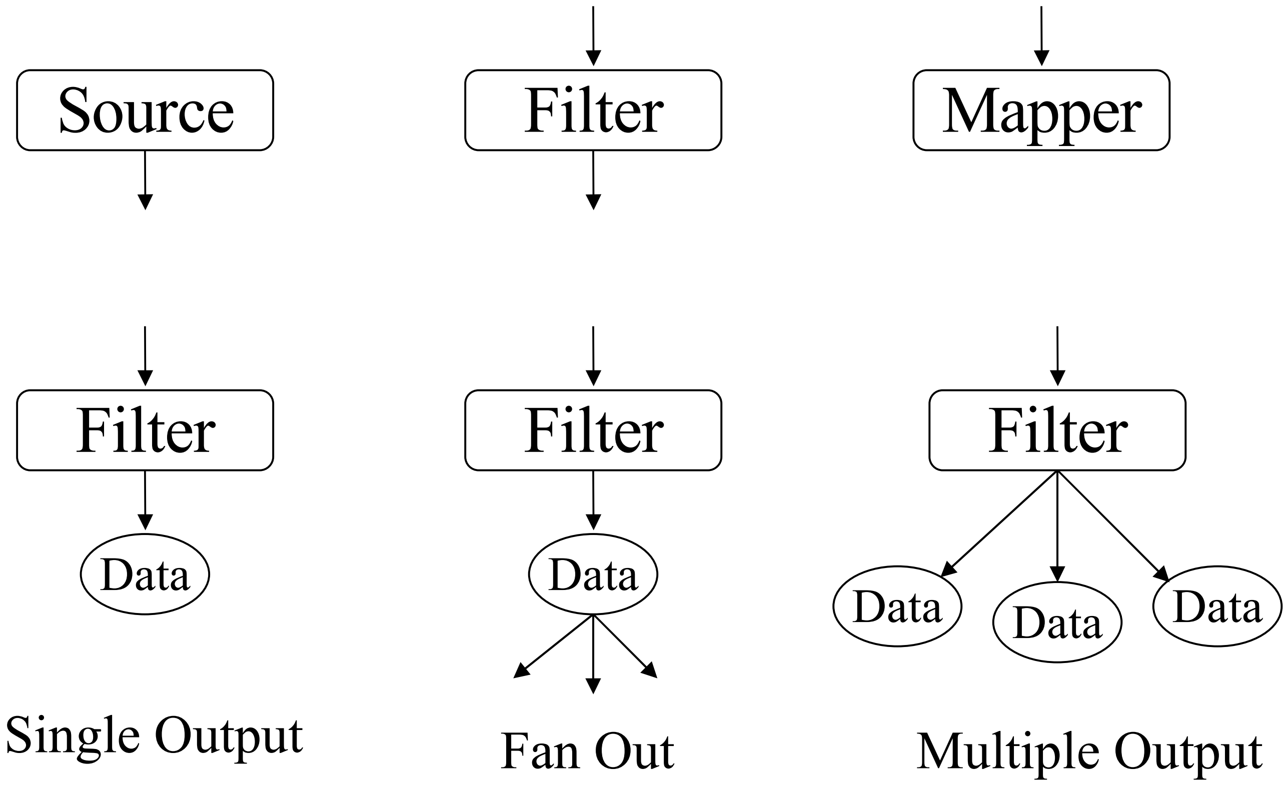


Multiplicity: number of inputs / outputs supported

Multiplicity of connections - 1

- Two special multi-connection cases:
  - **Fan out**
    - one module supplies the same data to many other modules
      - 1 output : N module connectivity
  - **Multiple outputs**
    - one module producing a number of different outputs that connect to different modules
      - N outputs : N module connectivity

Multiplicity of connections - 2

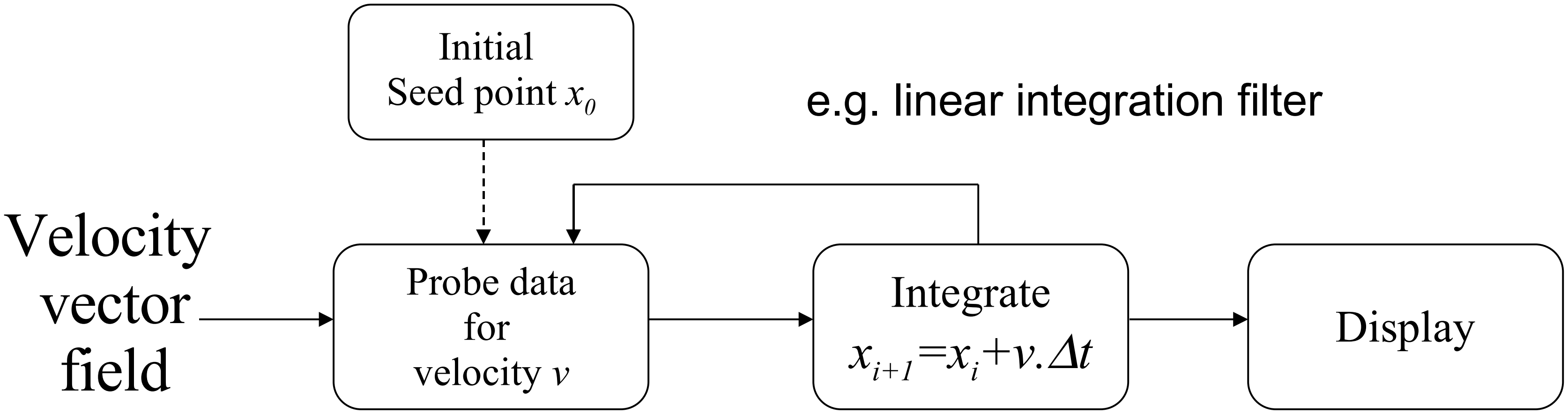


- Multiplicity allows the consideration of **parallel processing** in the visualisation pipeline
  - useful for “real-time” type demands on large data sets

Loops in pipeline/*network*

- Pipelines so far = acyclic graphs
  - no loops

- Loops will be needed, especially for visualisation of simulation data



Initial Seed point x_0

e.g. linear integration filter

Velocity vector field

Probe data for velocity v

Integrate $x_{i+1}=x_i+v.\Delta t$

Display

Visualisation Pipeline : Execution Control

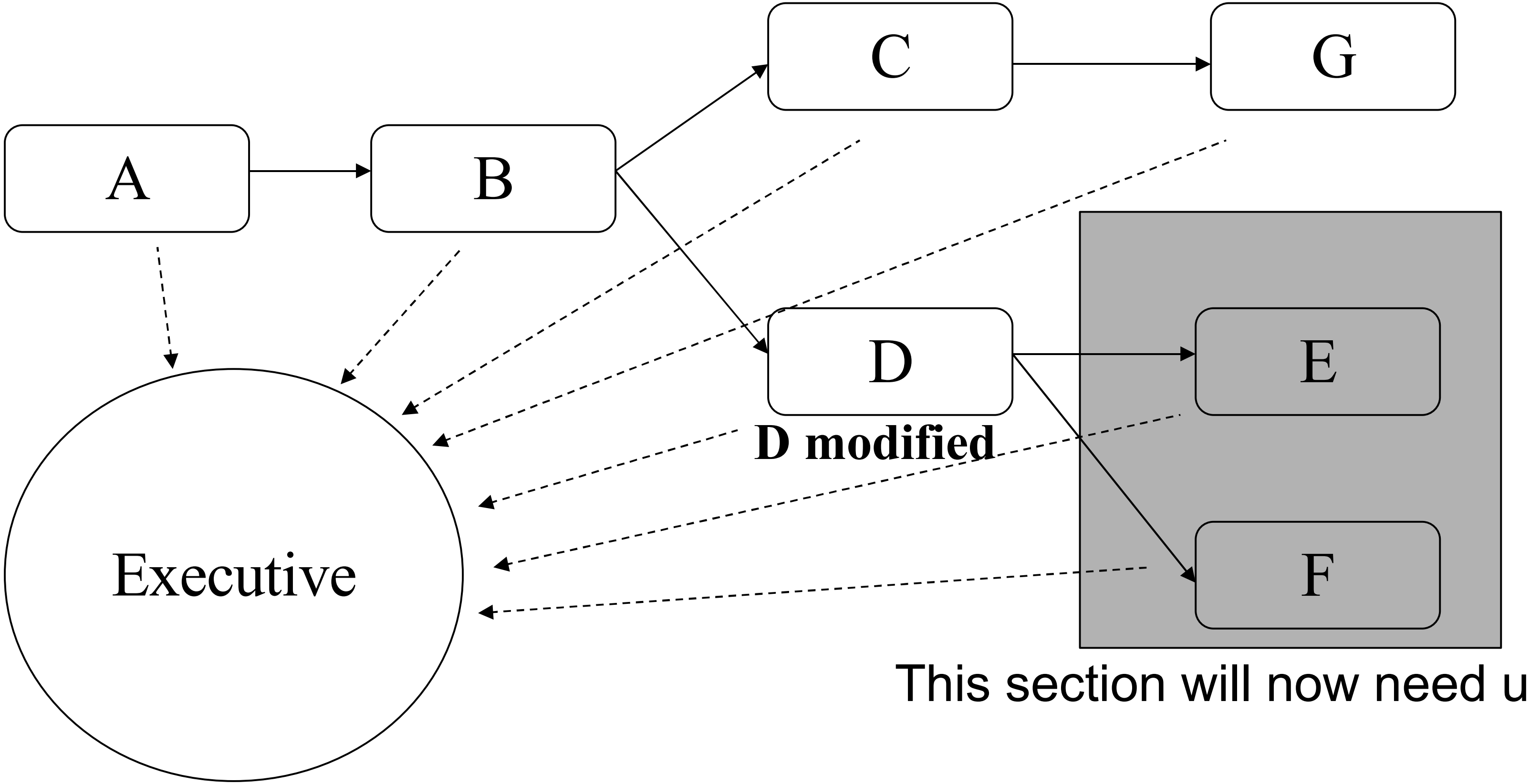
- **Problem:** *ensuring all parts of pipeline are up to date if a parameter is modified by user, and ensure synchronisation is maintained?*

- Solutions:

1. **Event-Driven:**
    - centralised **executive** (i.e. controller) notes change occurrences and re-executes effected modules
2. **Demand-Driven**
    - when output is requested by a mapper object, the **network is re-executed starting with source objects**

Event-driven Execution Control

- **Explicit Control** of Execution by executive

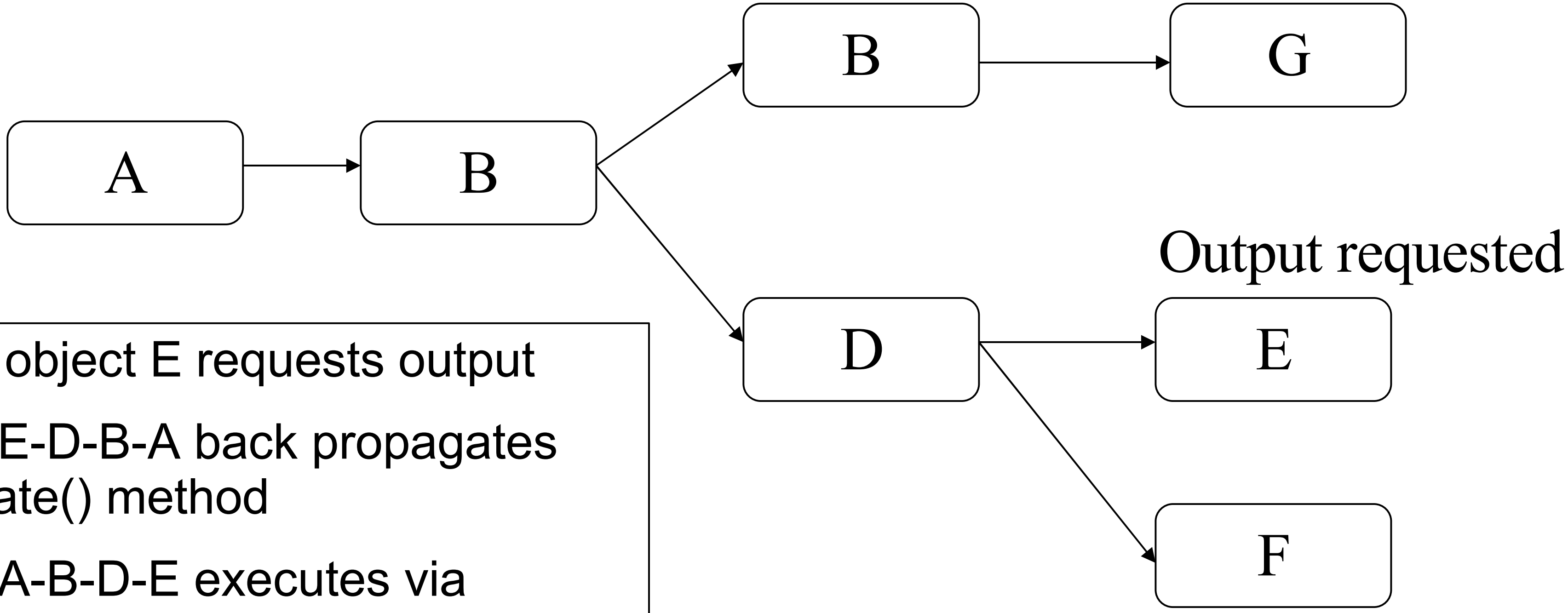


This section will now need updating

- **Advantages:** only update required *modules (i.e. objects)*
- **Disadvantages:** complexity of control, updates even without user demand,

  update called for every trivial modification, even if it’s not needed, making a series of modifications re-executes the network multiple times.

Demand-driven Execution Control



Mapper object E requests output

- Chain E-D-B-A back propagates via Update() method
- Chain A-B-D-E executes via Execute() method.

- **Implicit Control** of Execution by pipeline dependency tree
- **Advantages:** simplicity, no global knowledge required
- **Disadvantages:** inefficient to re-execute module if nothing has changed

Execution Control - Methods

- **Event-driven**
  - control of modules is **explicit** by executive
  - pipeline is always up to date (even if not required)
  - distribute modules (tasks) across computers
- **Demand-driven**
  - control of modules is **implicit** by user/module requests
  - simple, no central point of control/failure
  - do not re-execute a module unless required for output

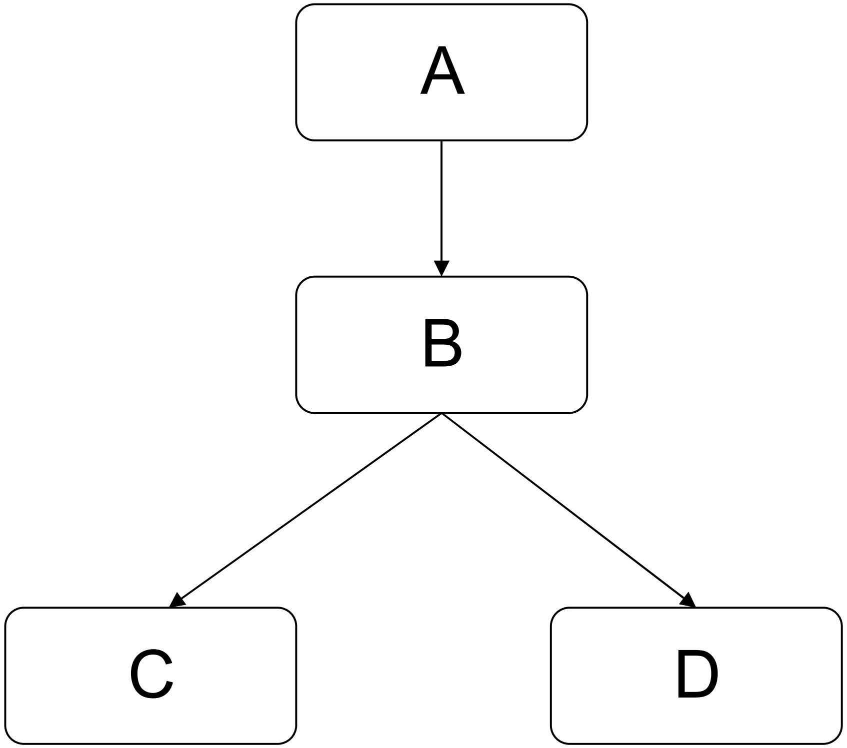
Memory and *re*-computation trade-off - 1

- **Problem** : do we store intermediate results in the pipeline ?
- **Yes** ⇒ keep memory allocated ⇒ **static memory model**
  - memory intensive, **beware** of large datasets
  - **saves** computation
- **No** ⇒ release memory allocation ⇒ **dynamic memory model**
  - module may need to be re-executed
  - computation intensive, **beware** of slow processors (or large data sets too!)
  - **saves** memory

Memory and *re*-computation trade-off - 2

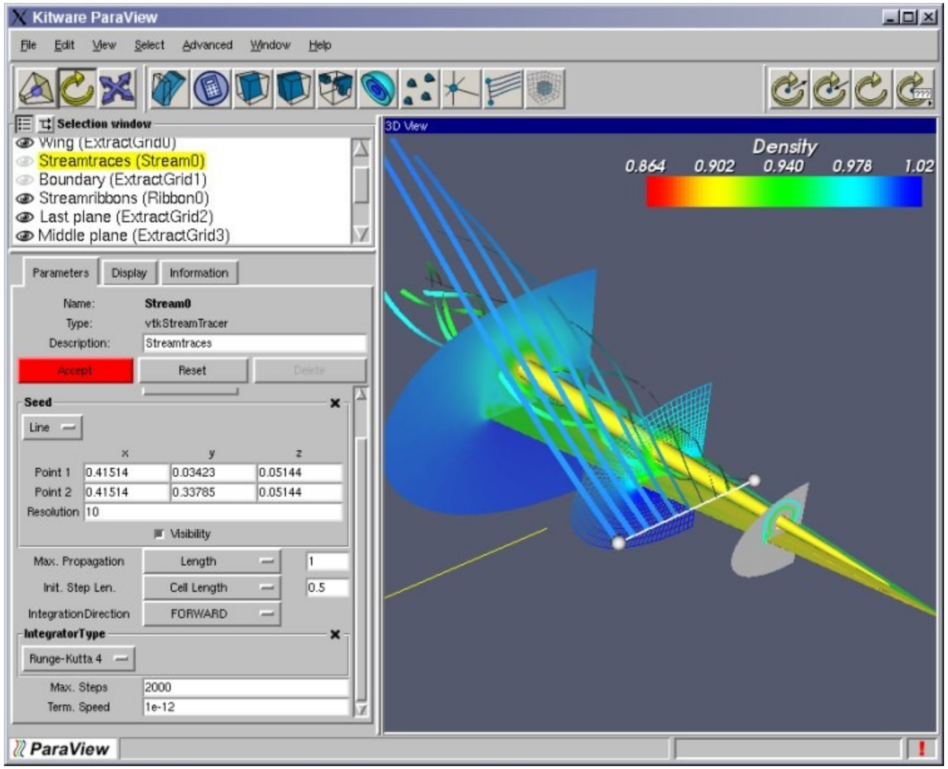
Modules are dependent on results from previous modules

A & B execute twice with a dynamic memory model if C and D execute and **once with a static memory model**



Best solution : dependency analysis

Visualisation packages



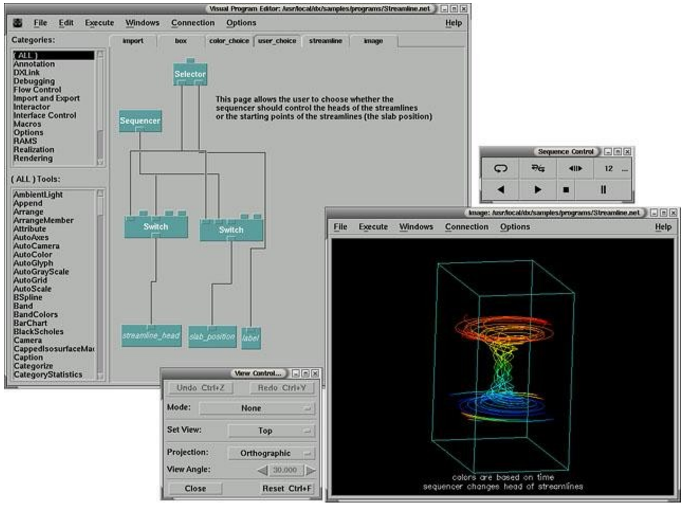
Name: Stream0
Type: vtkStreamTracer
Description: Streamtraces

Seed: Line

	x	y	z
Point 1	0.41514	0.03423	0.05144
Point 2	0.41514	0.33785	0.05144
Resolution	10		

Visibility

Max. Propagation: Length 1
Init. Step Len.: Cell Length 0.5
IntegrationDirection: FORWARD
IntegratorType: Runge-Kutta 4
Max. Steps: 2000
Term. Speed: 1e-12



This page allows the user to choose whether the sequencer should control the heads of the streamlines or the starting points of the streamlines (the slab position)

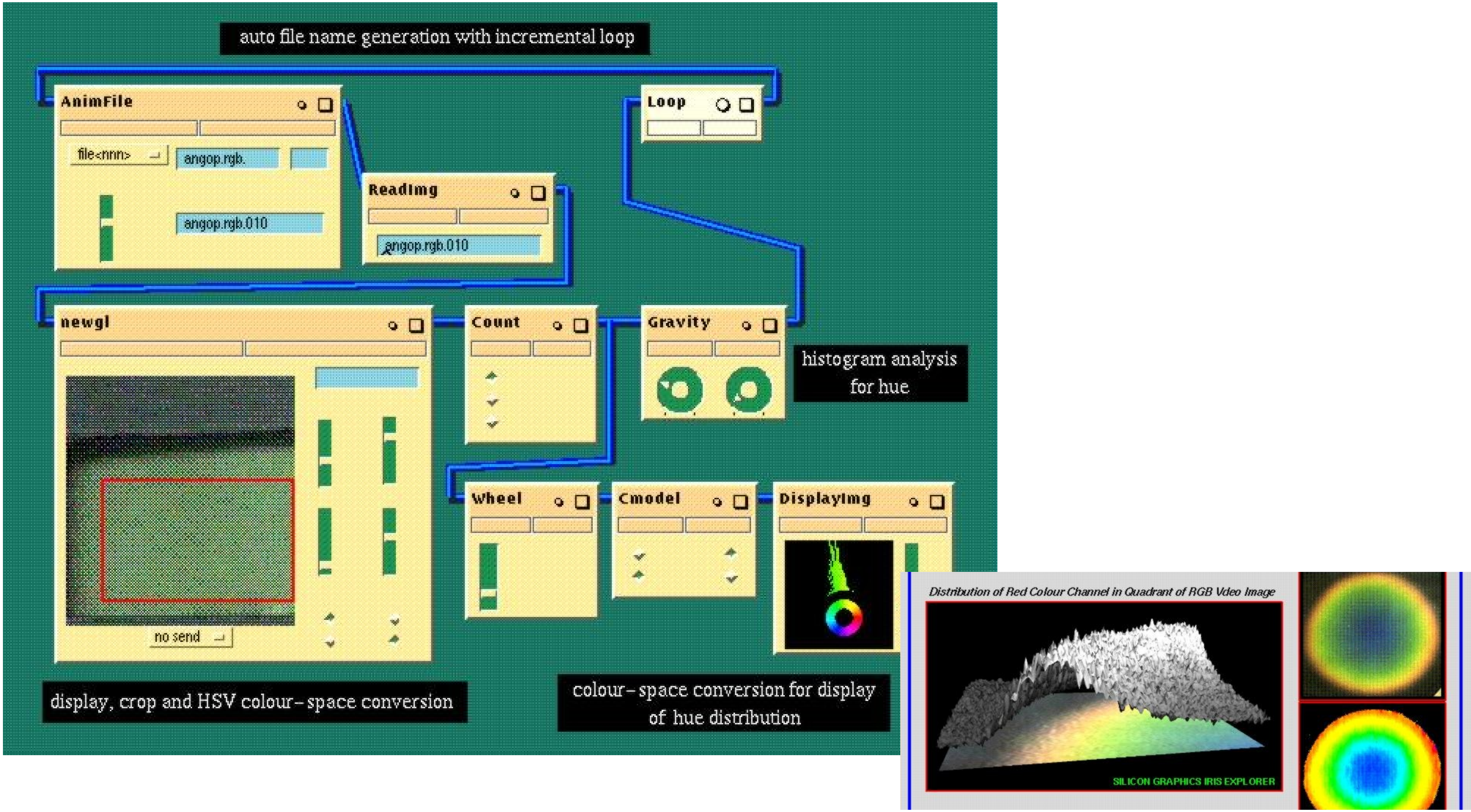
Mode: None
Set View: Top
Projection: Orthographic
View Angle: 30.000

colors are based on time
sequencer changes head of streamlines

- AVS : www.avs.com
- Paraview : www.paraview.org (free)
- OpenDX : http://www.opendx.org (free)

- Generally designed for less specialist users
  - in terms of visualisation techniques
  - **extensibility limited** to available macro/interface languages (often visual based)

Visual Programming



- Iris Explorer : Mechanical Engineering http://people.bath.ac.uk/enprgp/

Visualisation Pipeline : in VTK

- Each module is a VTK object *(C++/TCL/Java)*
  - Connect modules together by using:
    - `SetInput()`
    - `GetOutput()`
    - e.g. to connect modules A and B so B takes as input the output of B
      - TCL: `A setInput [ B GetOutput]`
      - Java: `A.SetInput( B.GetOutput() );`
- **VTK pipeline**
  - **demand-driven execution control** maintained **implicitly**
  - **memory** for intermediate results can be explicitly controlled
    - by default (**static** model)

Summary

- The Visualisation **Pipeline**
  - pipeline **connectivity**
  - pipeline **execution**
    - Overall – *An **Architecture for Visualisation***