Visualisation

UG4 / M.Sc. Course – 2008

Taku Komura
tkomura@inf.ed.ac.uk

Institute for Perception, Action & Behaviour

Course Outline

- **18 Lectures**
  - lecture notes on-line (http://www.inf.ed.ac.uk/teaching/courses/vis/)
  - background reading (mainly on-line, also textbook)
- **2 Assessed Practicals**
  - 2 programming tasks
  - Visualisation Toolkit – VTK
  - prior weekly practicals introducing VTK
- **Assessment**
  - 1.75 hour examination (70%)
  - 2 practical assignments (15% each)
    - (variation between UG4 and M.Sc. requirements)

VTK – The Visualisation Toolkit

- VTK is a C++ library (toolkit) that implements:
  - visualisation data structures
  - visualisation algorithms
  - common data interfaces (import / internal / export)
  - visualisation pipeline
  - visual output in OpenGL rendering
- Additional features
  - interfaces in TCL, python & Java (N.B. Java via JNI)
  - open source and platform independent

VTK – Practicals / Software

- **All practicals in VTK**
  - assessed assignments (use TCL, Java or C++)
    - TCL advised and will be presented
    - Java / C++ will have limited support
  - weekly non-assessed practicals
    - ~20 minutes per week
    - based in TCL
- **Software**
  - installed on DICE
  - available for home use: http://www.kitware.com/vtk
  - pre-build linux RPMs (from DICE) – see course homepage

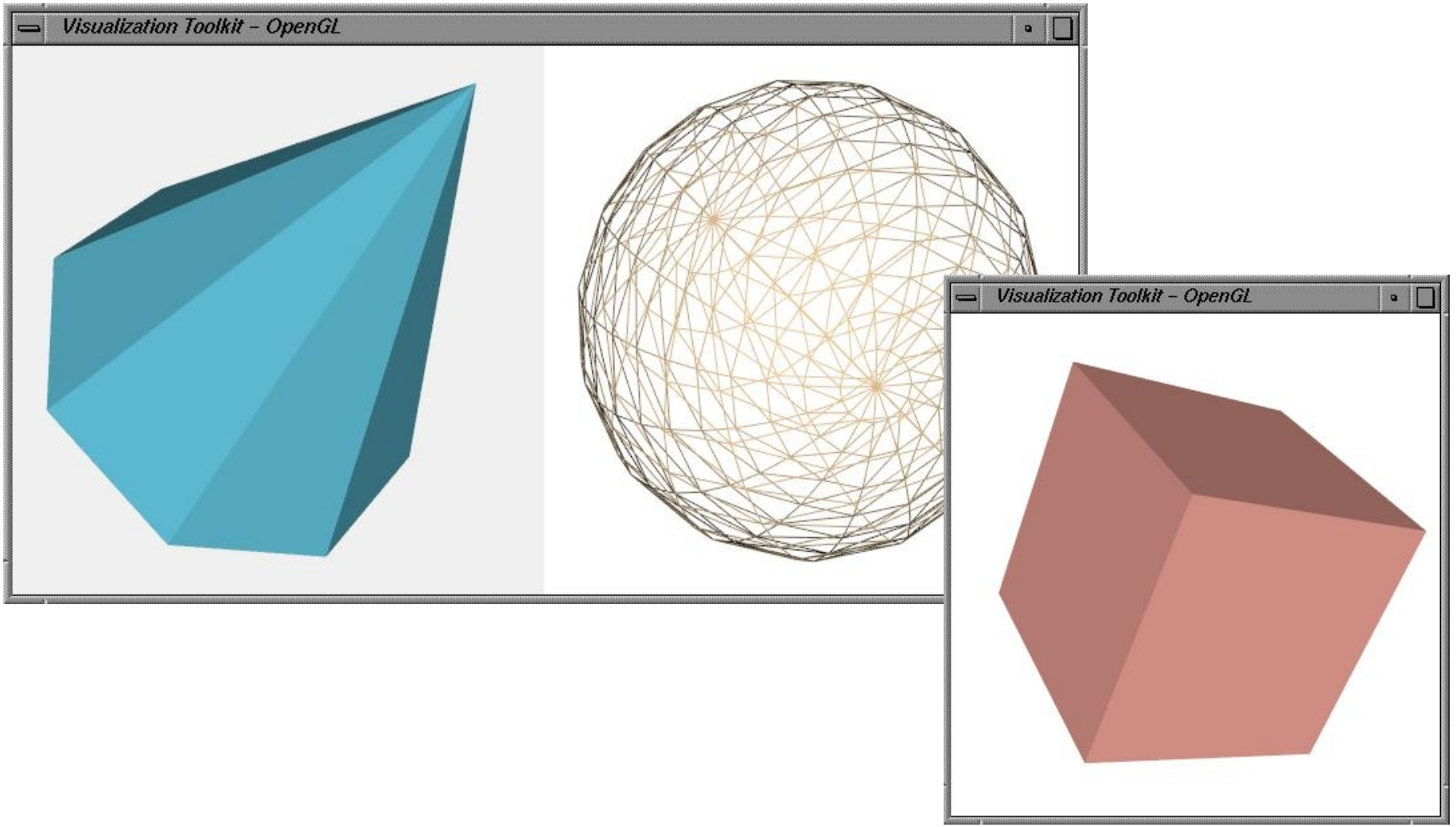
VTK : in summary

- Our **provider of a computer graphics architecture** for visualisation
  - VTK is a set of methods (toolkit) that implement a variety of visualisation operations
  - Implements a **visualisation pipeline**
  - Platform independent (we use linux, DICE)
  - **Object-orientated visualisation**
  - Program in C++ or Java or use an interpreted language such as Tcl/Tk or Python
  - VTK also implements basic tools for visualisation:
    - **3D computer graphics output & basic interactive user input**

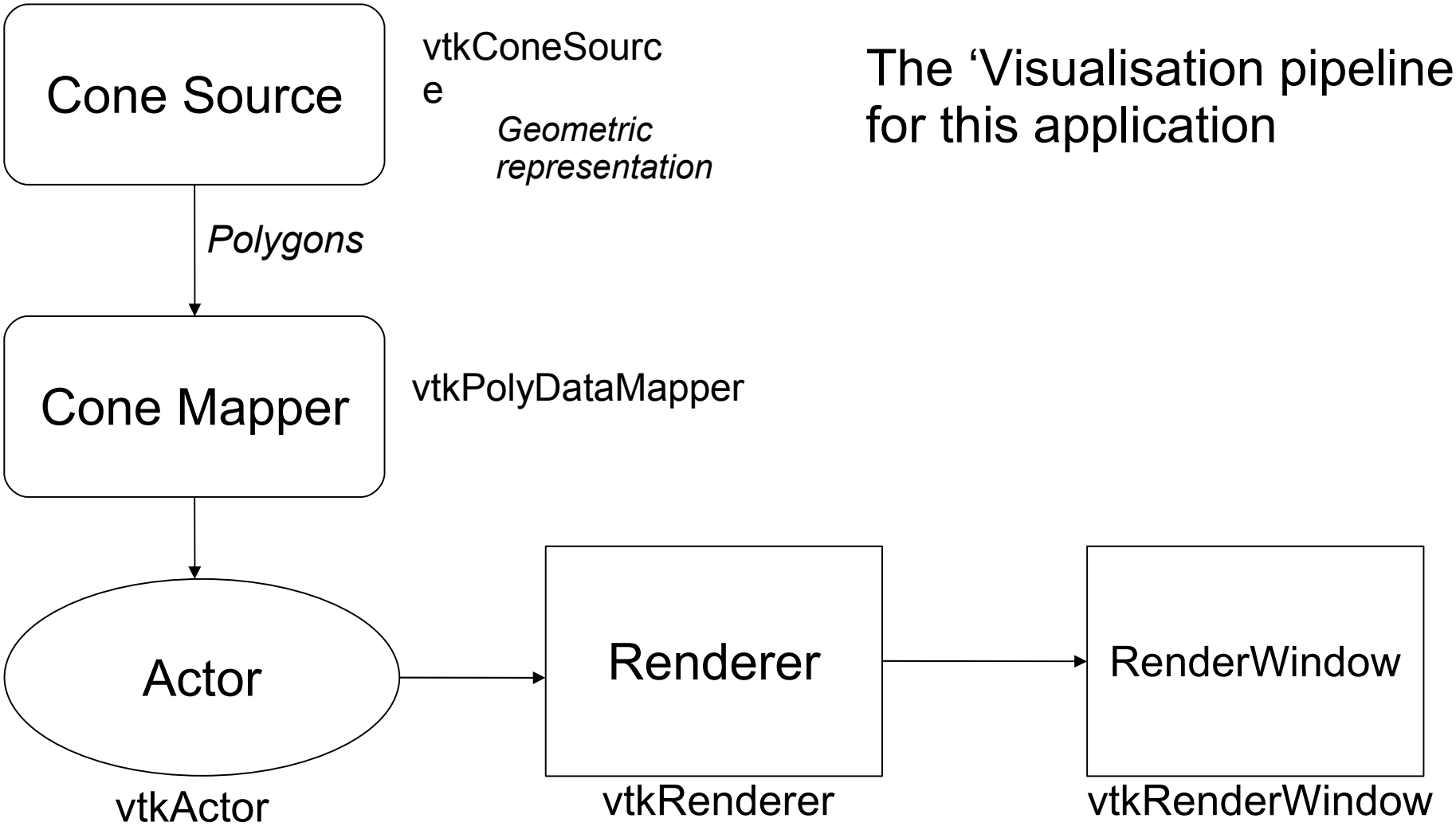
Computer Graphics Objects in VTK

- To convert a data structure into graphical object in VTK, use an object called a ***mapper***
- Graphics objects in vtk are known as ***actors***
  - Controls graphics properties such as colour and shading
  - Position, rotation and surface properties also specified by actor methods
  - transformation from object to world co-ordinates
- Actors are rendered in the scene by the ***renderer object***
  - Controls camera and lighting properties
- The renderer draws to a ***render window*** object
  - Controls window size
  - Can display or capture to an image file

Graphical Objects in VTK



Example : drawing cone



The 'Visualisation pipeline' for this application

VTK Objects : TCL / Java

- TCL: Command with class name creates new object of that class
  - Java : `Object obj = new Object();`
  - Tcl : `Object obj`
  - *VTK is object-orientated; TCL itself is not*
  - *A note on tcl/tk (tickle-talk), tcl/vtk .....*
    - *TCL (Tool Command Language) is a dynamically allocated interpreted programming language*
    - *Commonly used for GUI application with GUI toolkit TK - tcl/tk*
    - *Here we are doing **visualisation** (rather than GUI) so we use VTK – although not generally known as tcl/vtk !*

Drawing a cone : TCL

```
# create a rendering window and renderer
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1

# create a cone geometry source object
vtkConeSource cone
  cone SetResolution 8

# create mapper object and map cone
   geometry
vtkPolyDataMapper coneMapper
  coneMapper SetInput [cone GetOutput]
```

```
# create an actor object and set
# mapper
vtkActor coneActor
  coneActor SetMapper coneMapper

# assign our actor to the renderer
ren1 AddActor coneActor

# render scene
renWin Render
```


Drawing a cone : Java

```
public class Cone {
  public static void main (String []args) {
    // create an instance of vtkConeSource
    vtkConeSource cone = new vtkConeSource();
    cone.SetHeight( 3.0 );
    cone.SetRadius( 1.0 );
    cone.SetResolution( 8 );

    // create vtkPolyDataMapper and map cone source
    vtkPolyDataMapper coneMapper = new vtkPolyDataMapper();
    coneMapper.SetInput( cone.GetOutput() );
```


Drawing a cone : Java

```
    // create actor and assign mapper
    vtkActor coneActor = new vtkActor();
    coneActor.SetMapper( coneMapper );

    // create renderer and add actor
    vtkRenderer ren1 = new vtkRenderer();
    ren1.AddActor( coneActor );

    // create render window and add renderer
    vtkRenderWindow renWin = new vtkRenderWindow();
    renWin.AddRenderer( ren1 );
}
```


Drawing a cone : Java *Boiler Plate Code*

```
// We import the vtk wrapped classes first.
import vtk.*;
// Then we define our class.
public class Cone {
  // In the static contructor we load in the native code (via JNI).
  // The libraries must be in your path to work.
  static {
    System.loadLibrary("vtkCommonJava");
    System.loadLibrary("vtkFilteringJava");
    System.loadLibrary("vtkIOJava");
    System.loadLibrary("vtkImagingJava");
    System.loadLibrary("vtkGraphicsJava");
    System.loadLibrary("vtkRenderingJava");
  }
```


TCL basics : variables

- Variables
  - Are all strings
  - Set using 'set *variable* value'
  - Reference using $*variable*
- *Dynamic arrays*
- Expression
  - Use *expr* to evaluate an expression
- Print results to standard output with *puts*
  - *useful for debugging*
- *Comments starts with #*

TCL basics : variables

```
# Compute the circumference of a circle

set pi 3.14159
set radius 2
set pos(0) 11
set pos(1) 12
set area [expr $radius * $pi * 2.0]
puts $area
```


TCL basics : loops

- for loop : 3 arguments : {start } {end} {every}

```
# Example to print number 1-10 and their squares
for {set num 1} {$num <= 10} {incr num} {
    set numsqr [expr $num*$num]
    puts "$num => $numsqr"
}
```

- while loop : 1 argument : {end condition}

```
# print numbers 1 to 10
set x 0
while {$x<10} {
    puts "x is $x"
    incr x
}
```

TCL basics : conditionals

- Exactly the same as C :

  if *boolean* then *body1* else *body2*

  - both *then* and *else* are optional

e.g. :

```
if {$x == 0} then {
puts "Only superheros, can divide by zeros!"
} else {
set slope [expr $y/$x]
}
```

Special Features of TCL/VTK interpreter

- Special method : **ListMethods.**
  - Invoked in combination with an object name
  - Find out which methods the object has
  - Listed according to the inheritance hierarchy
- Special command : **ListInstances**
  - Invoked in combination with a class name.
  - Lists all instances of a particular class
- Special command : **DeleteAllObjects**
  - Clears the tcl/vtk interpreter for another session

VTK : interaction

- Create a new **vtkRenderWindowInteractor**
  - controls user interaction with VTK visualisation
  - **`vtkRenderWindowInteractor iren`**
- Set the RenderWindow object that it will control
  - **`iren SetRenderWindow renWin`**
- Make the interactor active and start processing events
  - **`iren Initialize`**
- Tcl code is still processed even though event loop entered

VTK : window interactor

- Functions available (**`vtkRenderWindowInteractor`**):
  - Rotate (left mouse button)
  - Zoom (Right mouse button)
  - Pan (left mouse + shift key)
  - ‘w’ Draw as a wireframe mesh
  - ‘s’ Draw as a surface mesh
  - ‘r’ Reset camera view
  - ‘u’ user defined command. Here, bring up window command box
    - `iren AddObserver UserEvent {wm deiconify .vtkInteract}`
  - ‘e’ exit
  - ‘p’ pick actor underneath mouse pointer

On-line Resources

- VTK
  - Manual: http://www.vtk.org/doc/release/5.0/html/
  - Examples: http://public.kitware.com/VTK/example-code.php
    - More examples: http://www.vtk.org/doc/release/5.0/html/pages.html
  - Everything else: http://www.vtk.org/
- **TCL**
  - Manual: http://www.tcl.tk/man/tcl8.4/TclCmd/contents.htm
  - Online tutorial: http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html
  - Everything else: http://www.tcl.tk/
- Software: see course web page (linux) or http://www.vtk.org/
  - N.B. DICE versions - vtk : 5.0

Summary

- VTK
  - Overview of **VTK rendering pipeline**
  - simple example in TCL and Java
  - basis of **TCL programming language**
  - **VTK interactive visualisation**