Computer Graphics: Introduction to the Visualisation Toolkit

Visualisation – Lecture 2

Taku Komura

Institute for Perception, Action & Behaviour

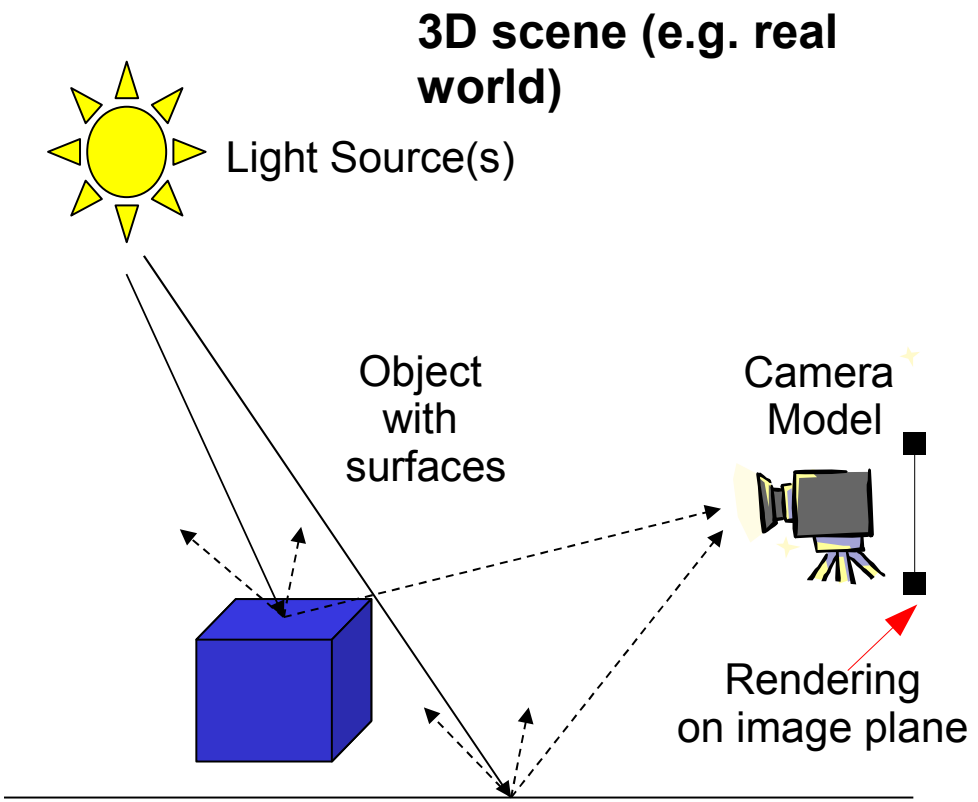
Last lecture .....

- **Visualisation** can be greatly enhanced through the use of **3D computer graphics**
  - computer graphics are our **tool in visualisation**

- In order to do effective visualisation we need:
  - to know some computer graphics *(this lecture)*
  - a computer graphics architecture *(VTK)*

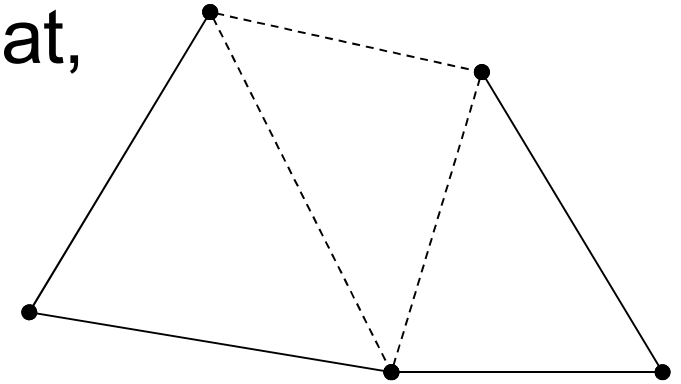
Computer Graphics : simulation of light behaviour in 3D

- Effective simulation require to model:
  - object representation **(geometry)**
  - object illumination **(lighting)**
  - camera model **(vision)**
    - *world to image plane projection*
    - ***rendering:*** *converting graphical data into an image*



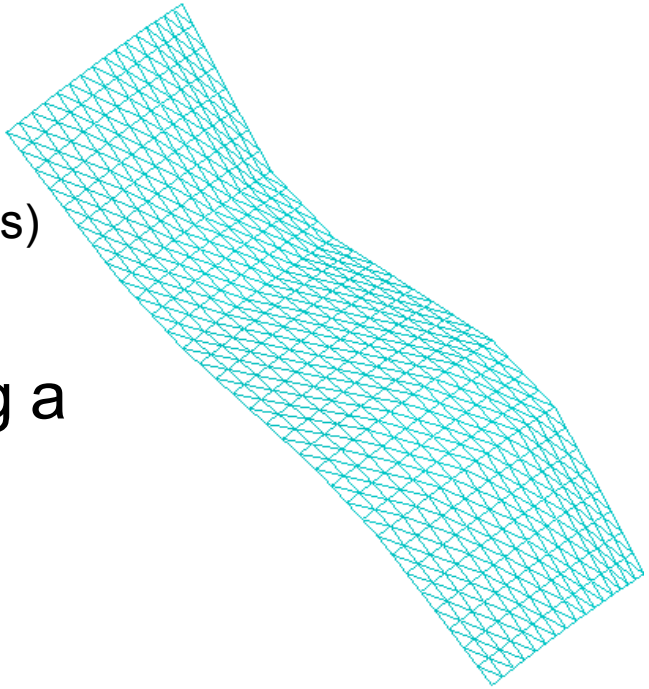
Data Representation : 3D shape

- Approximate smooth surfaces with flat, planar polygons
  - polygons formed of edges & vertices

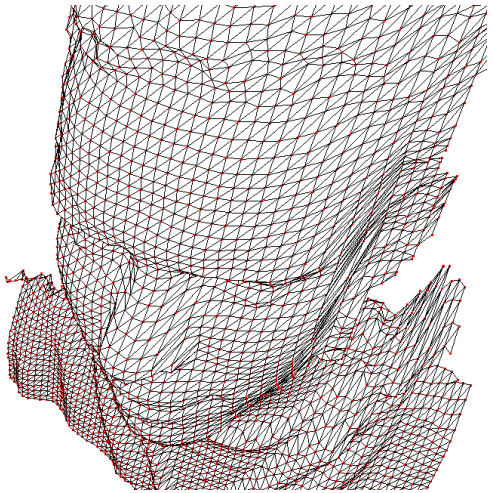


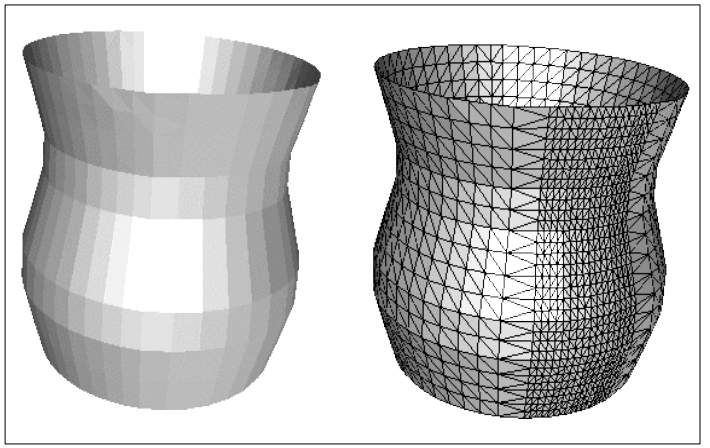
Hierarchy of Surface Representation

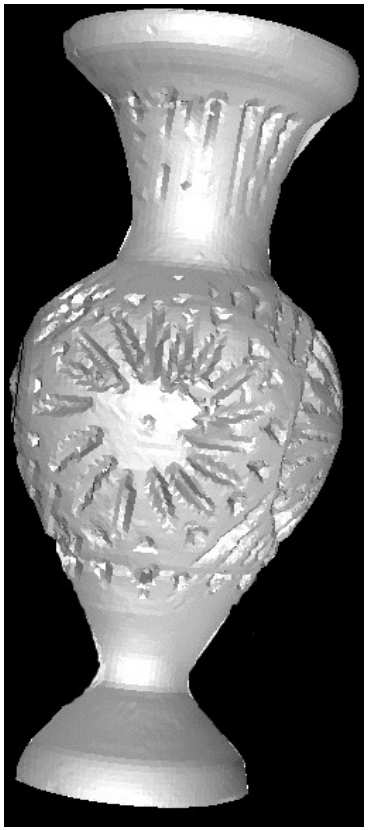
  - **vertex:** positional point (2D or 3D)
  - **edge:** joins 2 vertices
  - **polygon:** enclosed within N edges (cf. faces)
    - polygons share common edges
  - **mesh:** set of connected polygons forming a surface (or object)



Surface mesh : examples





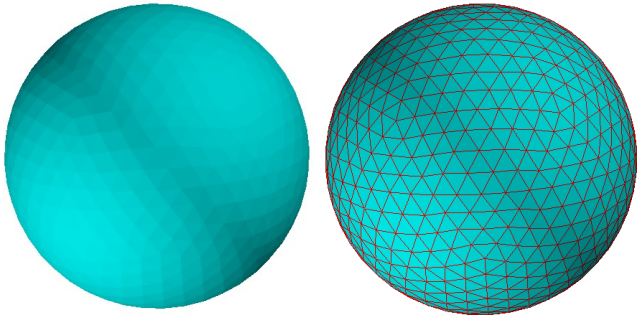


- **Different resolutions** alter perception of surface smoothness
- triangles generally fastest to draw
  - modern graphics cards : 20 - 225 million triangles per second

Mesh Based Representation

- 3D file formats:
  - set of vertices in $\mathbb{R}^3$
  - polygons reference into vertex set
    - implicitly define edges
  - e.g.

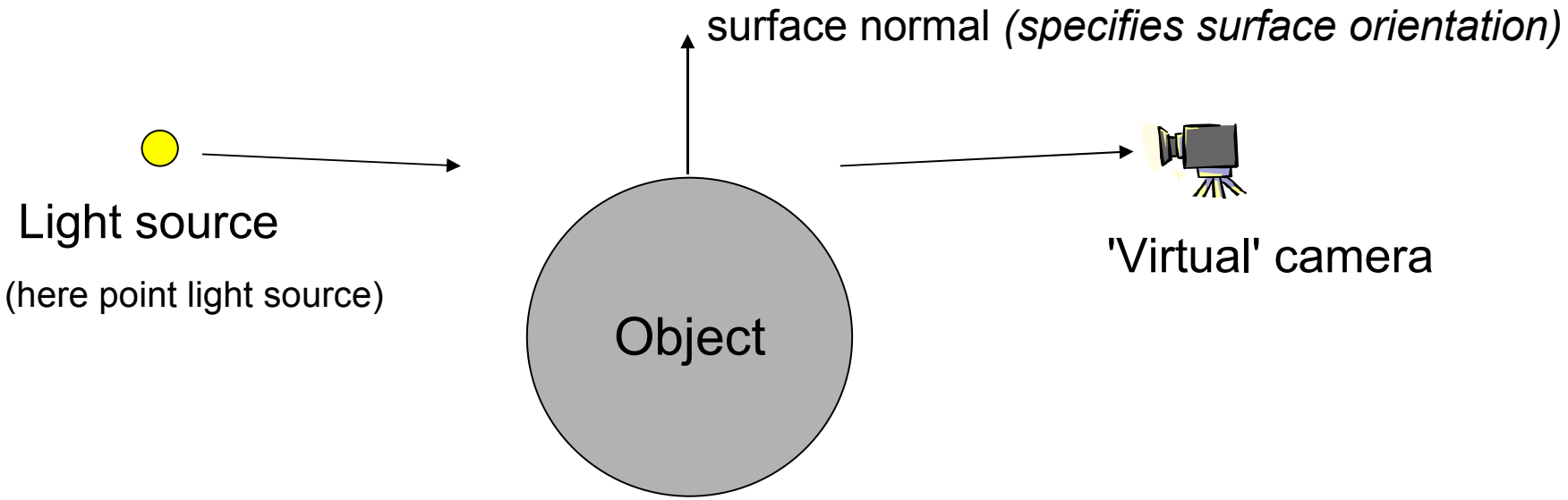
```
vertex 0 0 0
vertex 0 1 0
....
polyon 3 2 1 3
polyon 3 5 6 8
...
```



```
#VRML V1.0 ascii
#
Separator {
        Material {
                ambientColor 0.2 0.2 0.2
                diffuseColor 1.0 1.0 1.0
        }
        Coordinate3 {
                 point [
                        4.455030 -1.193380 1.930940,
                        4.581220 -1.506290 1.320410,
                        4.219560 -1.875190 1.918070,
                        3.535530 1.858740 -3.007500,
                        3.793260 1.185430 -3.034130,
                        4.045080 1.545080 -2.500000,
                        3.510230 3.468900 0.803110,
                        3.556410 3.514540 0.000000,
                        3.919220 3.078210 0.405431,
                       ....
                       ....
        IndexedFaceSet {
                 coordIndex [
                        0, 1, 2, -1,
                        3, 4, 5, -1,
                        6, 7, 8, -1,
                        9, 10, 11, -1,
                        12, 13, 14, -1,
                        15, 16, 17, -1,
                        18, 19, 20, -1,
                        21, 22, 23, -1,
                       ....
                       ....
```

Light interaction with surfaces

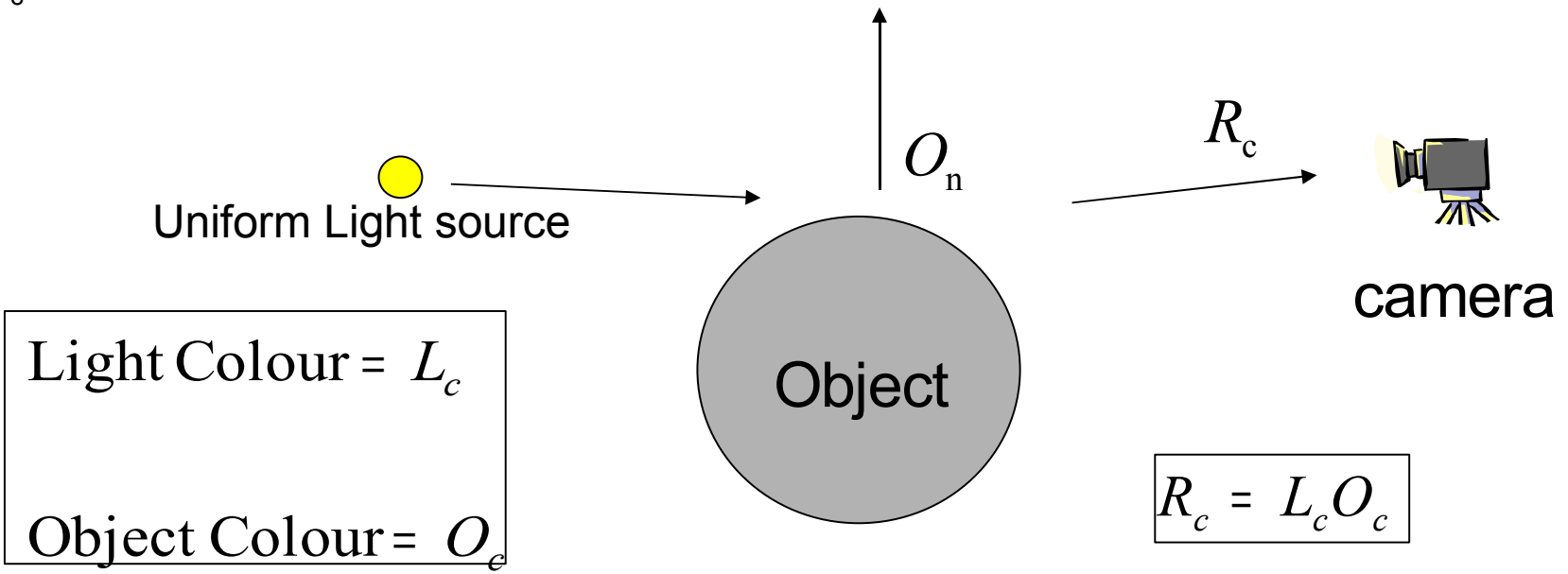
- Simple 3 parameter model
  - The sum of 3 illumination terms:
    - **Ambient :** 'background' illumination
    - **Specular :** bright, shiny reflections
    - **Diffuse :** non-shiny illumination and shadows



Ambient Lighting

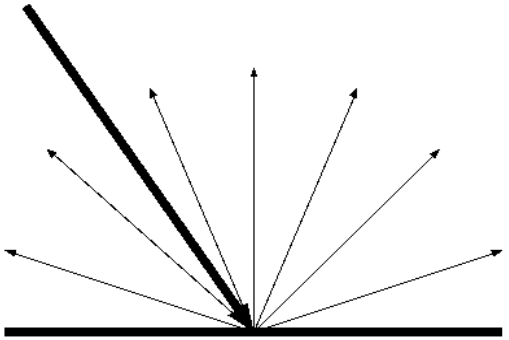
- light reflected or scattered from other objects
- simple approximation to complex 'real-world' process
- Result: **globally uniform colour for object**
  - R_c = resulting intensity
  - L_c = light intensity
  - O_c = colour of object

Example: sphere

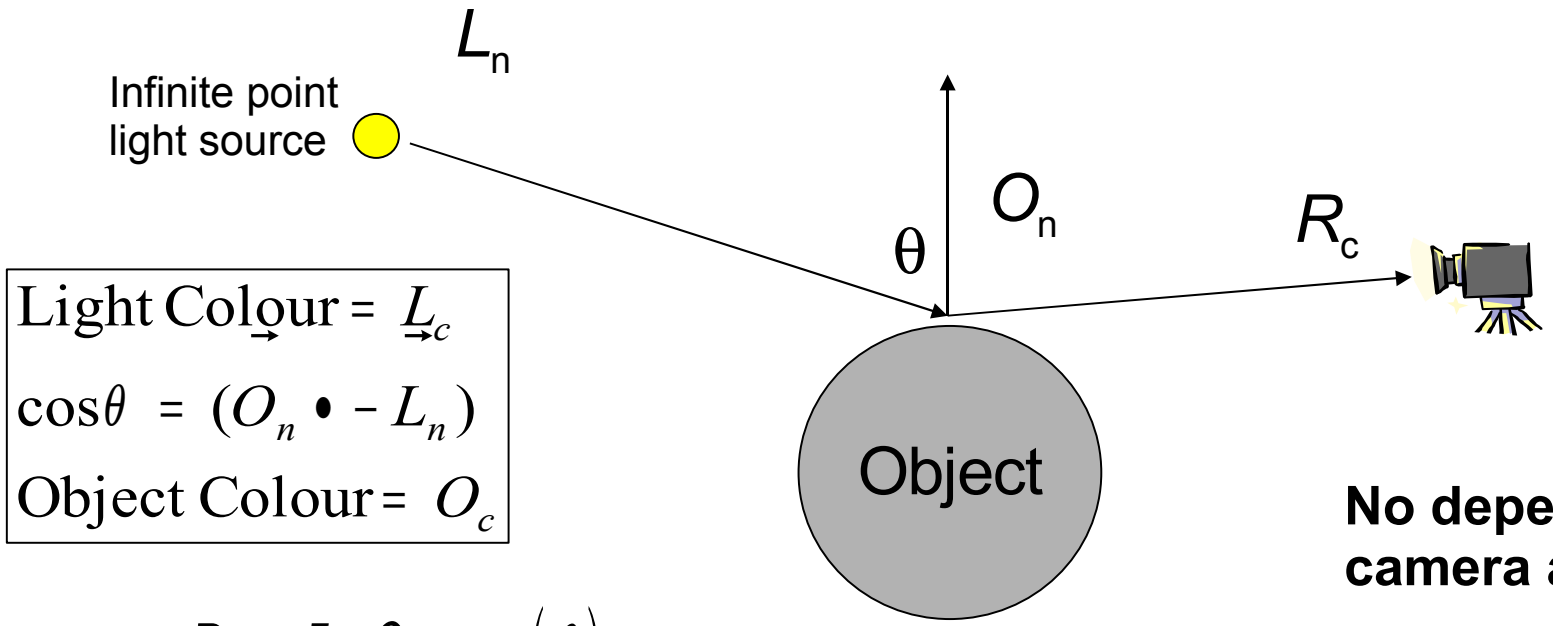


Diffuse Lighting

- The light scattered to all directions
  - considers the angle of incidence of light on surface
    (angle between light and surface normal)
  - Result: **lighting varies over surface with orientation to light**



Example: sphere (lit from left)



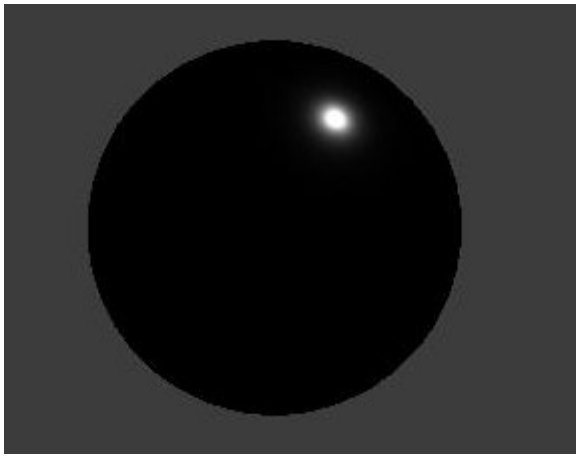
$$\text{Light Colour} = \underrightarrow{L_c}$$
$$\cos\theta = (O_n \bullet -L_n)$$
$$\text{Object Colour} = O_c$$

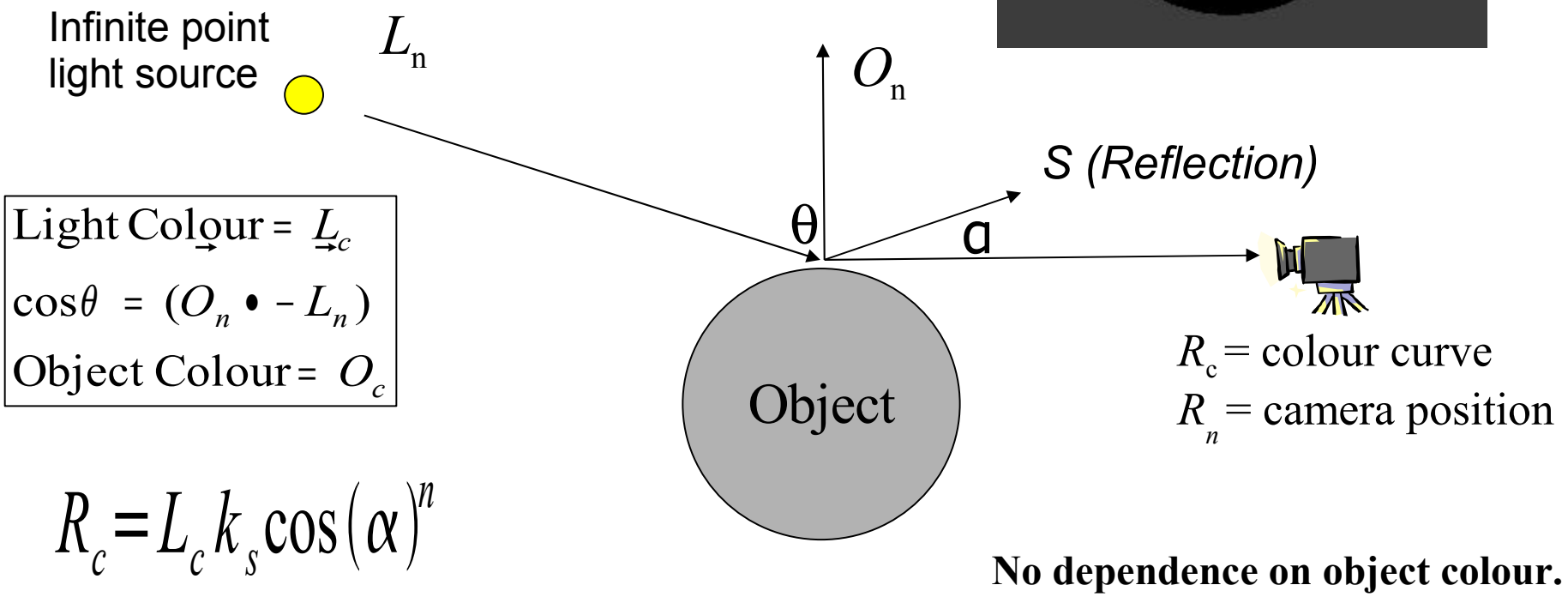
No dependence on camera angle!

$$R_c = L_c O_c \cos(\theta)$$

Specular Lighting

- Direct reflections of light source off shiny object
  - specular intensity *n* = shiny reflectance of object
  - Result: **specular highlight on object**





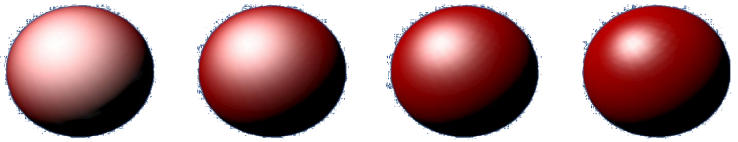
$$\text{Light Colour} = \vec{L_c}$$
$$\cos\theta = (O_n \bullet -L_n)$$
$$\text{Object Colour} = O_c$$

$$R_c = L_c k_s \cos(\alpha)^n$$

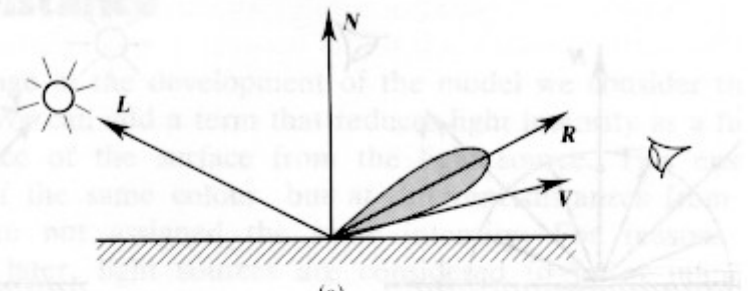
No dependence on object colour.

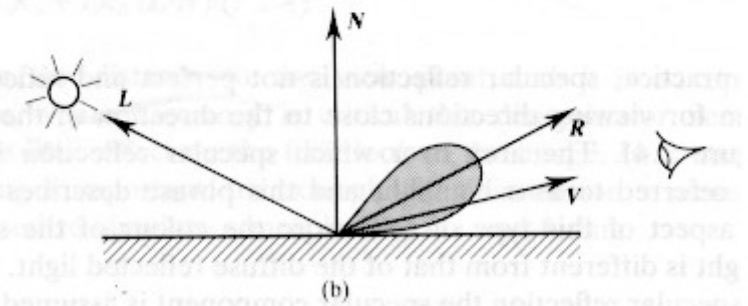
Specular Lighting

- Specular light with different *n* values



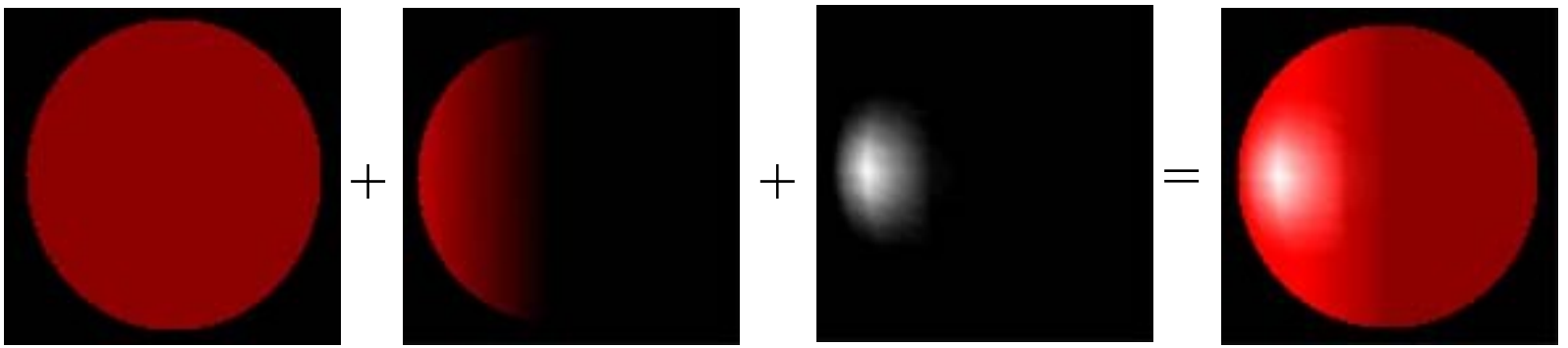
n=4 *n*=100





Combined Lighting Models

- $R_c = w_a(\text{ambient}) + w_d(\text{diffuse}) + w_s(\text{specular})$
  - for relative weights w_a, w_d, w_s
  - also specular power *n*

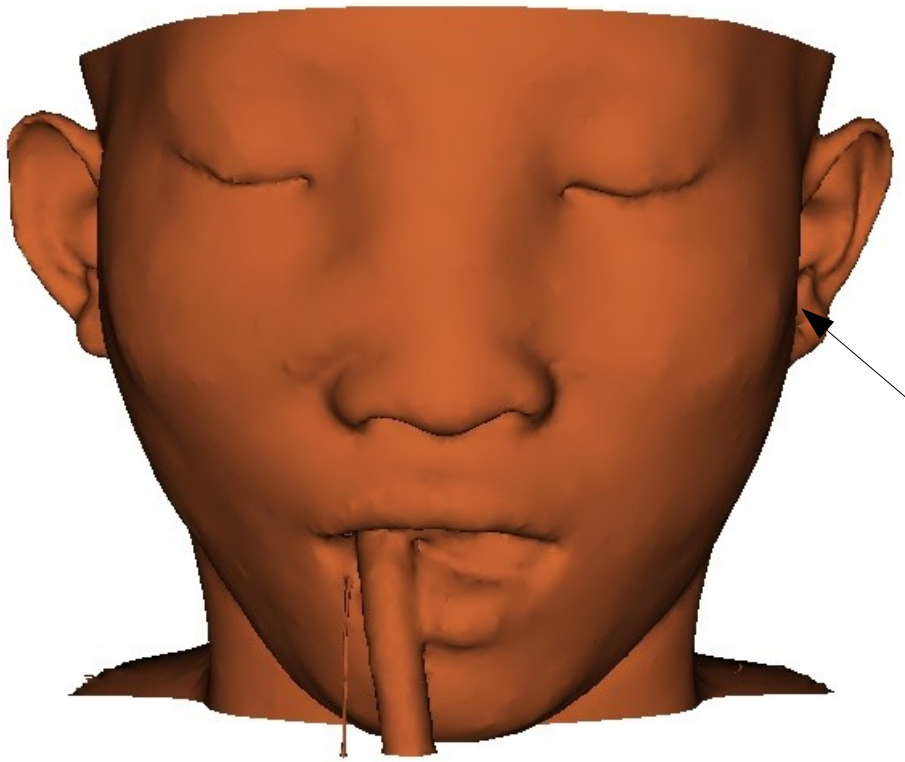


Ambient (colour) | Diffuse (directional) | Specular (highlights) | R_c

Demo Applet

- http://www.cs.unc.edu/~clark/courses/comp14-spr04/c
- http://www.cs.auckland.ac.nz/~richard/research-topics/

Surface Shape Perception - 1



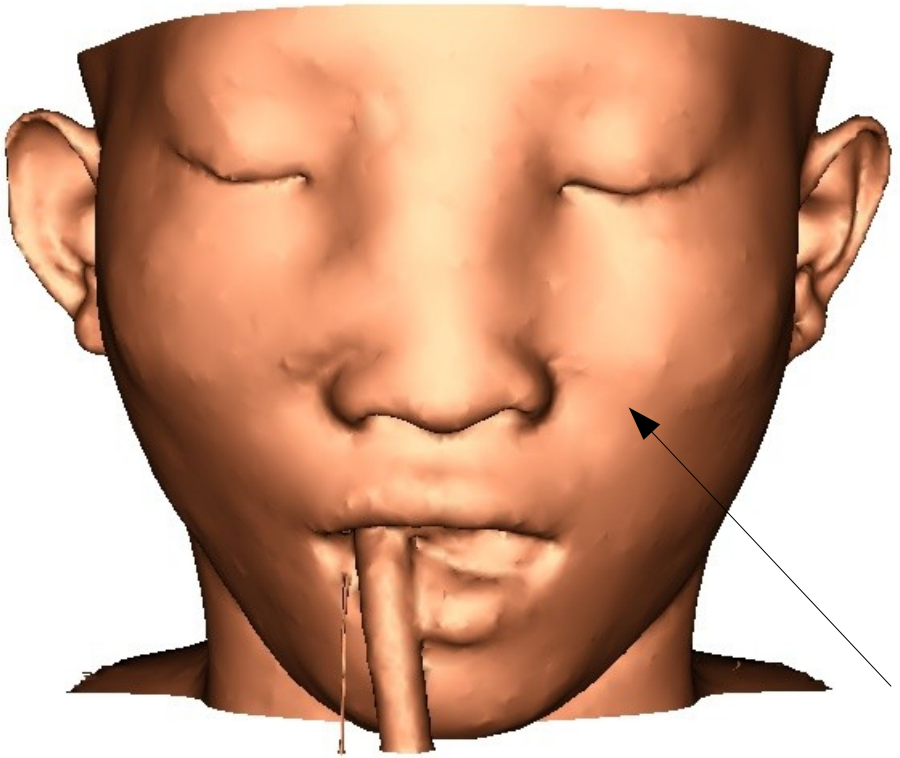
3D surface of the skin from a medical scanner.

Diffuse lighting only.

Light is coming from the top front

Perpendicular to light

Surface Shape Perception - 2

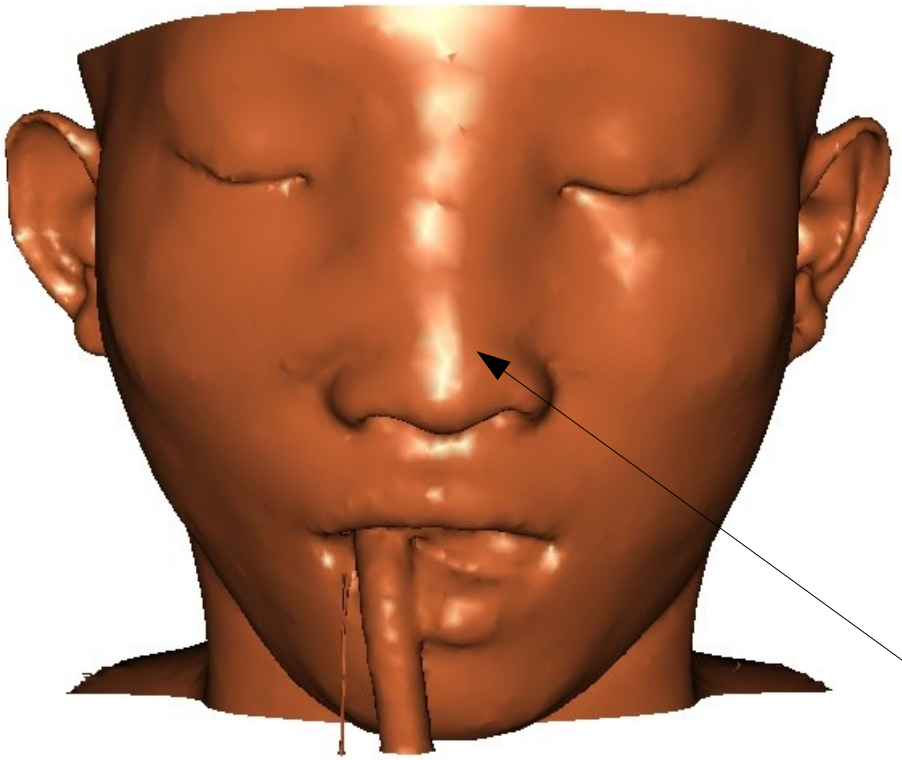


3D surface of the skin from a medical scanner.

Diffuse + specular lighting.

Specular Power = 4.0

Surface Shape Perception - 3



3D surface of the skin from a medical scanner.

Diffuse + specular lighting.

Specular Power = 200.0

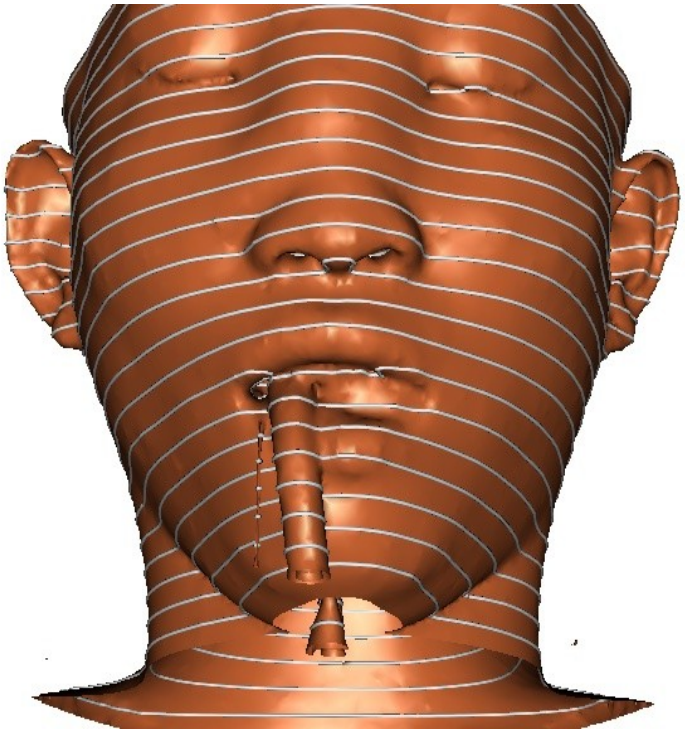
Edge of highlight

Perception of Shape

- Specular highlights
  - improve perception of surface shape features (e.g. nose)
  - ... but only where the highlight occurs

Other cues to shape

- **Interaction**
  - Ability to rotate the shape is extremely powerful.
  - Allows perception of edge contours (silhouette)
- **Texture**
  - The motion/direction of lines or patterns on the surface of the shape
- **Stereo**
  - Viewing depth with 2 eyes
  - Stereo displays frequently used for visualisation
  - Display 'feels' more 3D



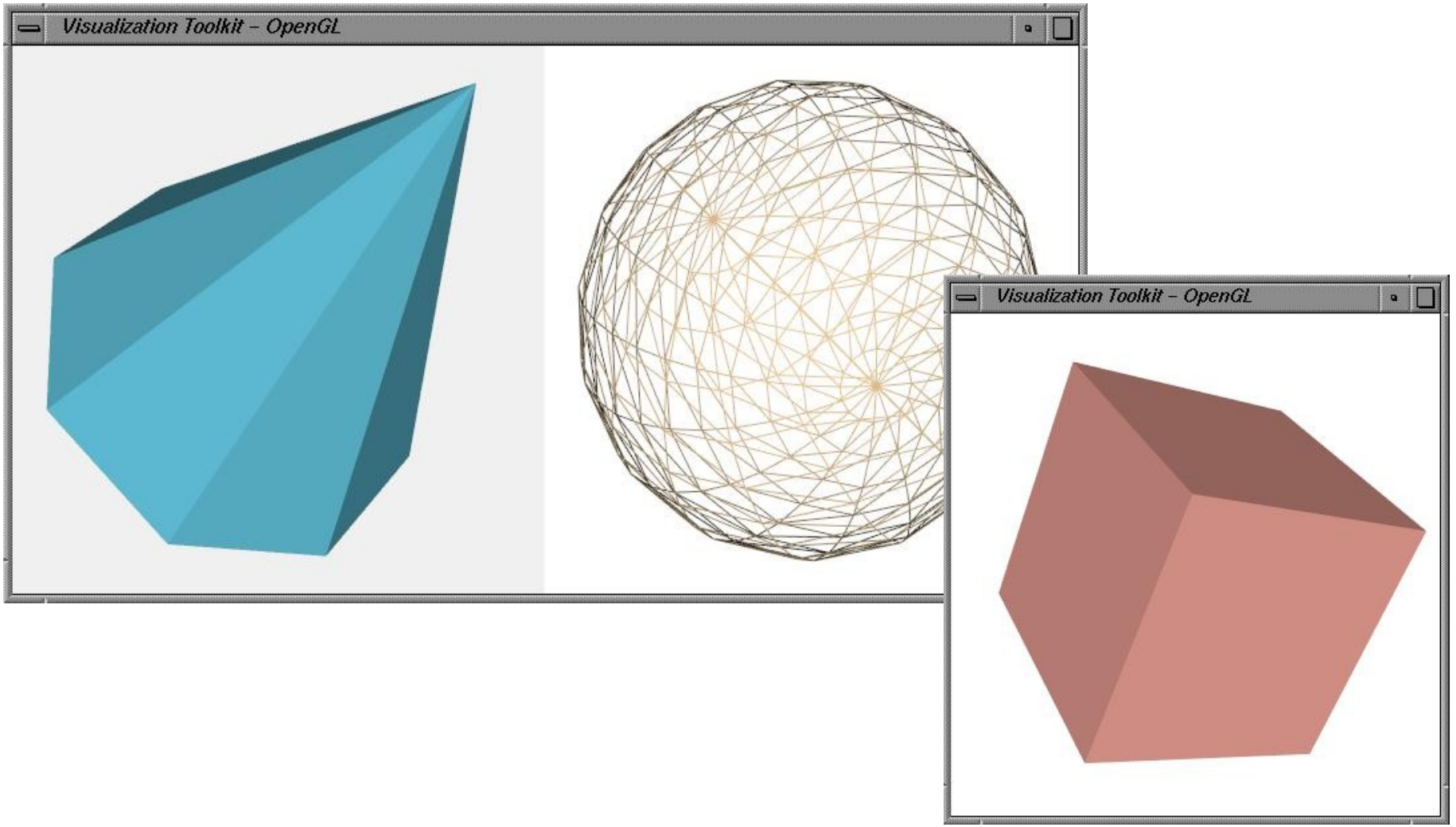
VTK : in summary

- 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**

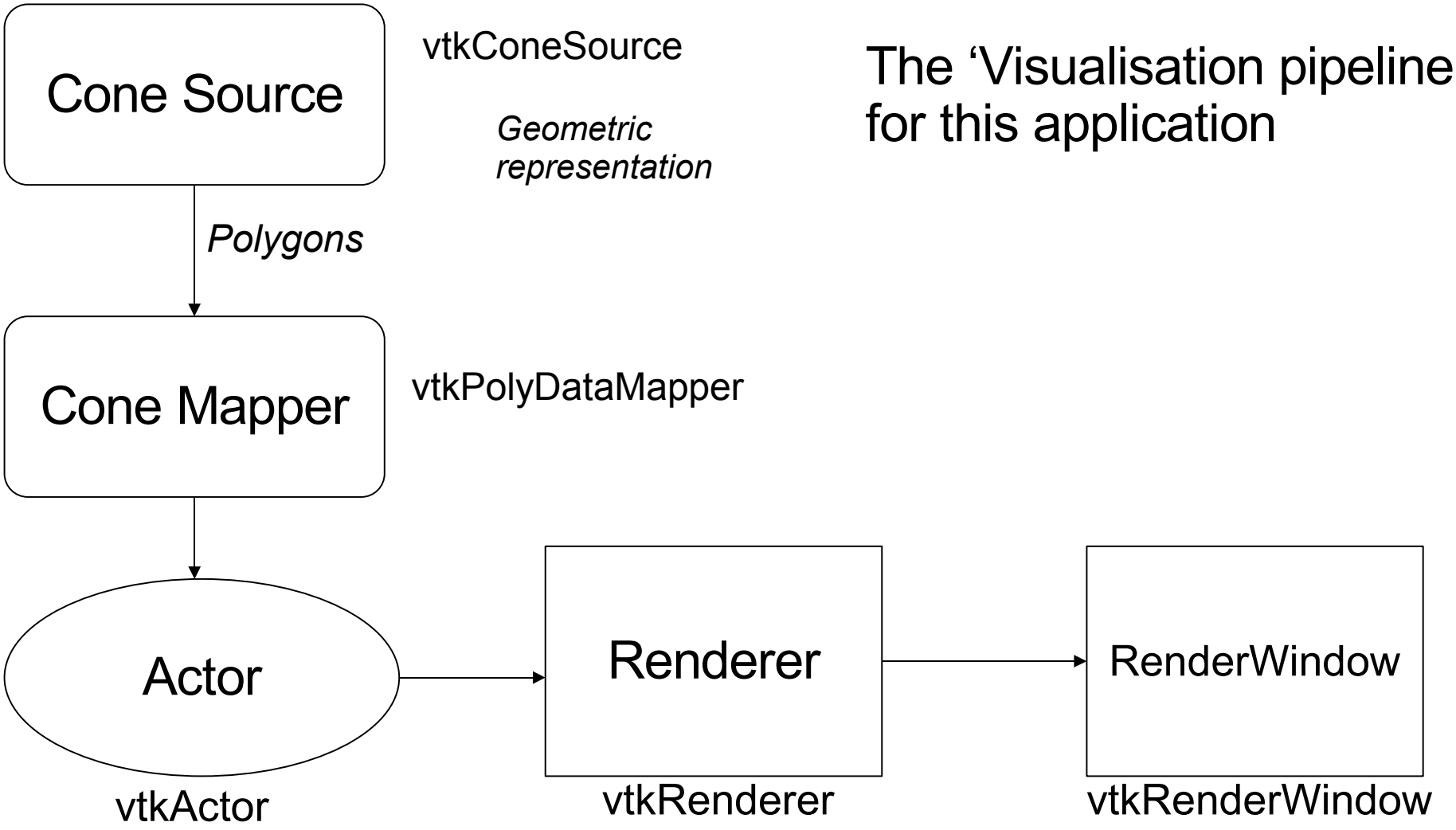
Computer Graphics Objects in VTK

- 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 : TCL Wrapper Code

```
# load vtk package and vtk user interaction command package
package require vtk
package require vtkinteraction
........

 # CODE FROM PREVIOUS SLIDE


........
```


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 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 SetUserMethod {wm deiconify .vtkInteract}`
  - ‘e’ exit
  - ‘p’ pick actor underneath mouse pointer

On-line Resources

- VTK
  - Manual: http://www.vtk.org/doc/release/4.2/html/
  - Examples: http://public.kitware.com/VTK/example-code.php
    - More examples: http://www.vtk.org/doc/release/4.2/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 : 4.2.4 & tcl : 8.4.7-2

Summary

- Computer Graphics (basics)
  - representing object geometry as **polygon meshes**
  - **illumination models** (ambient, diffuse, specular)
  - **camera model & projection** (VTK)
- VTK
  - Overview of **VTK rendering pipeline**
  - simple example in TCL and Java
  - basis of **TCL programming language**
  - **VTK interactive visualisation**
- Next lecture: systems architectures for visualisation