Leow Wee Kheng
CS3249 User Interface Development

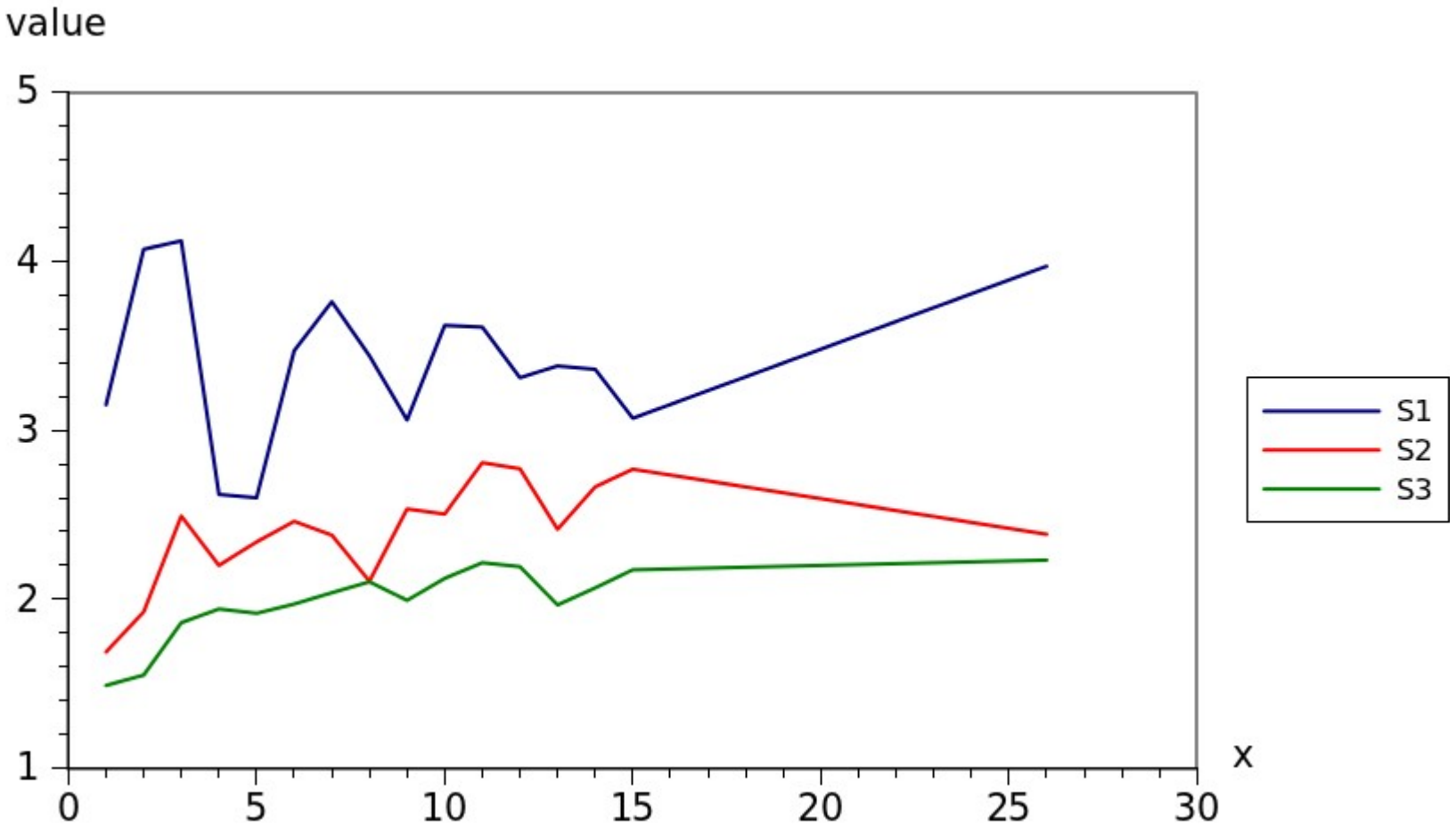
Qt with VTK (Part 1)

Data Visualisation

Humans are not good at reading numbers.

x	S1	S2	S3
1	3.15	1.69	1.49
2	4.07	1.92	1.55
3	4.12	2.49	1.86
4	2.62	2.2	1.94
5	2.6	2.34	1.92
6	3.47	2.46	1.97
7	3.76	2.38	2.04
8	3.44	2.11	2.1
9	3.06	2.53	1.99
10	3.62	2.5	2.12
11	3.61	2.81	2.22
12	3.31	2.77	2.19
13	3.38	2.41	1.97
14	3.36	2.66	2.07
15	3.07	2.77	2.17
26	3.97	2.38	2.23

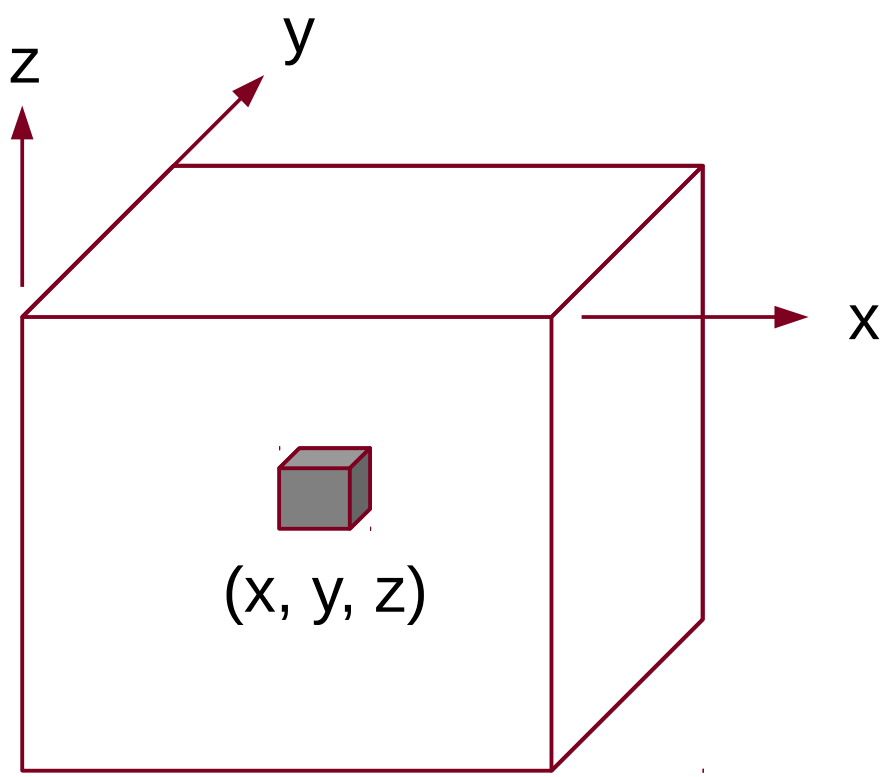
But we can understand pictures very well.



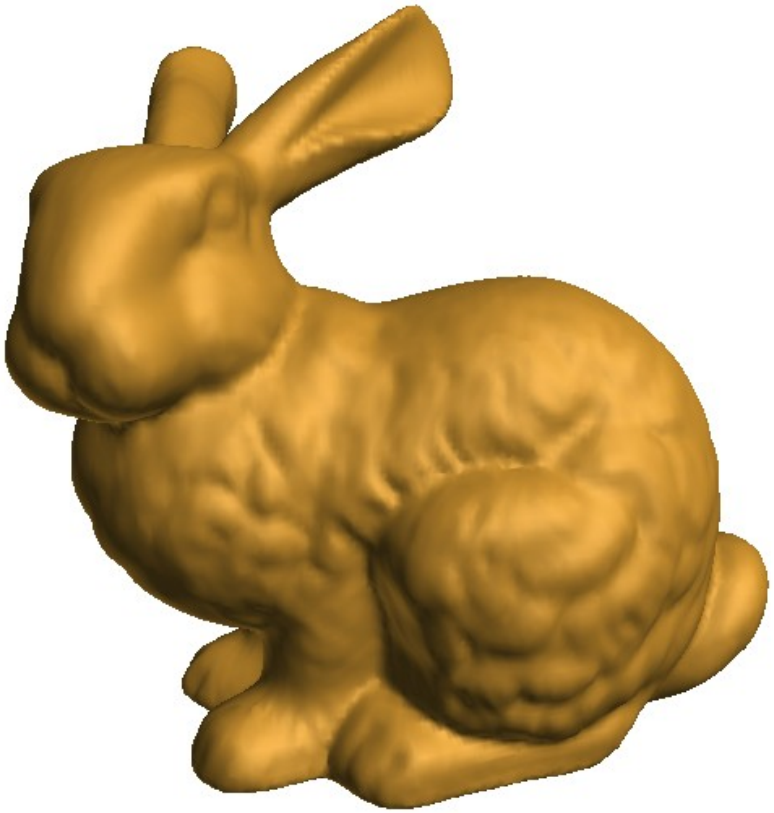
- Data can come in any number of dimensions.
- Some data have inherent structures:
  - image
    - 2D organisation of pixels.
    - Each pixel has 1 (grey) or 3 (colour) values.
  - file system
    - tree structure of nodes.
    - directory node can have sub-tree.
- We focus on images and 3D data.

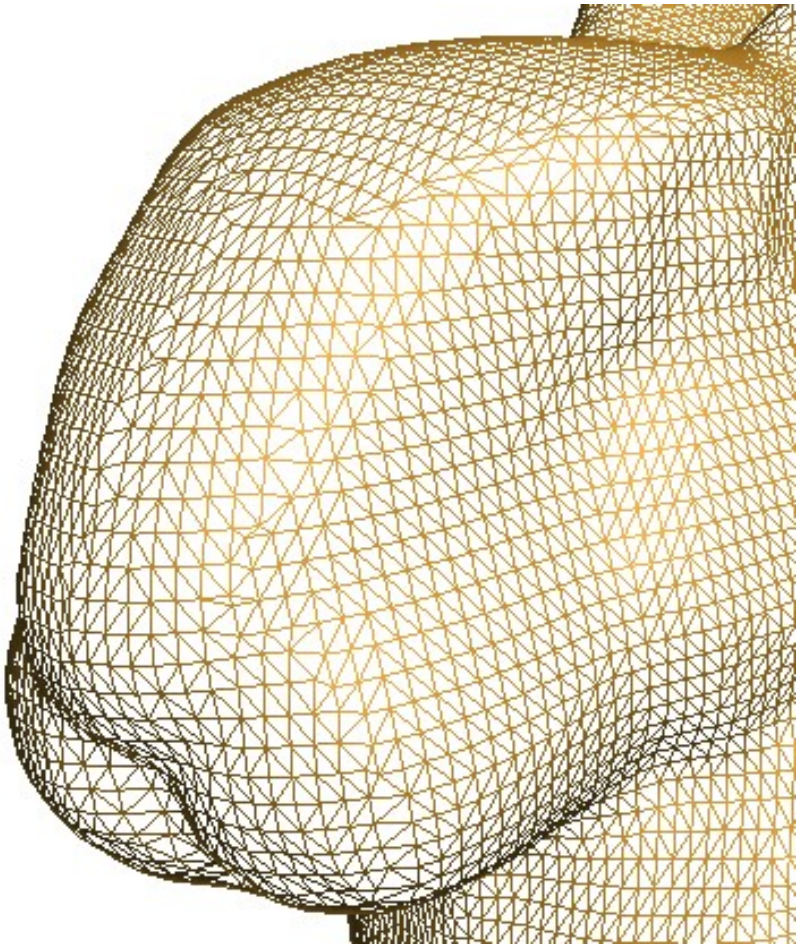
3D Data

- Volume data
  - Organise into 3D voxels (volume elements).
  - Each voxel is identified by 3 coordinates (x, y, z).
  - Each voxel contains an intensity (or colour) value.

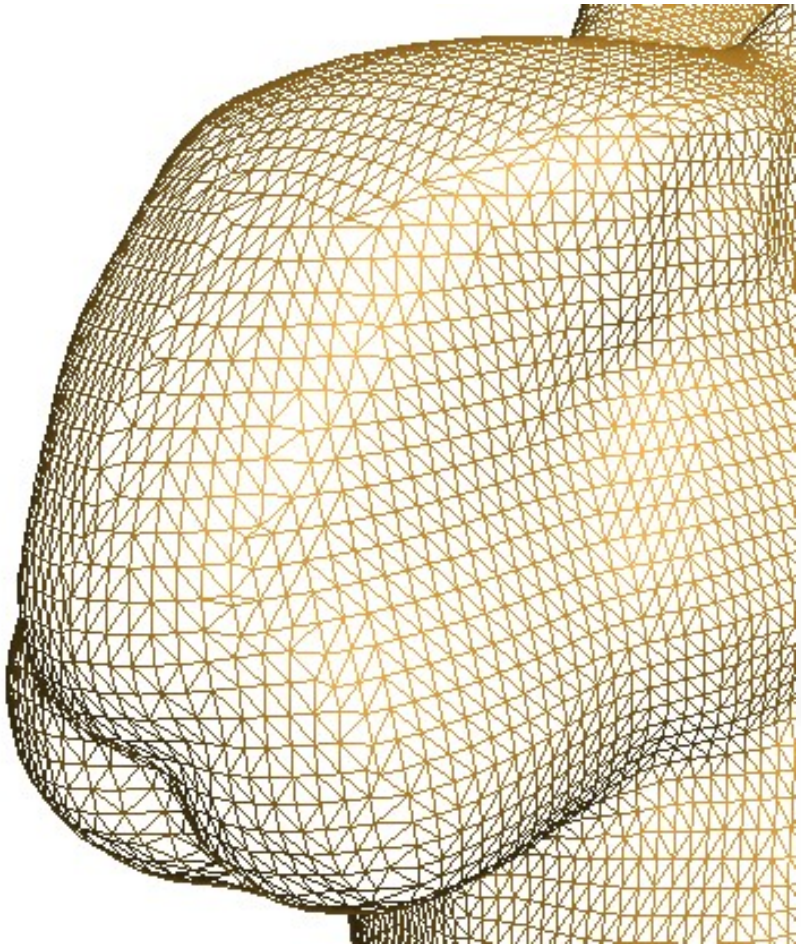


- 3D Mesh
  - Represents surfaces of 3D objects by vertices and edges.
  - Vertices are points on surfaces.
  - Edges connect vertices to form faces.

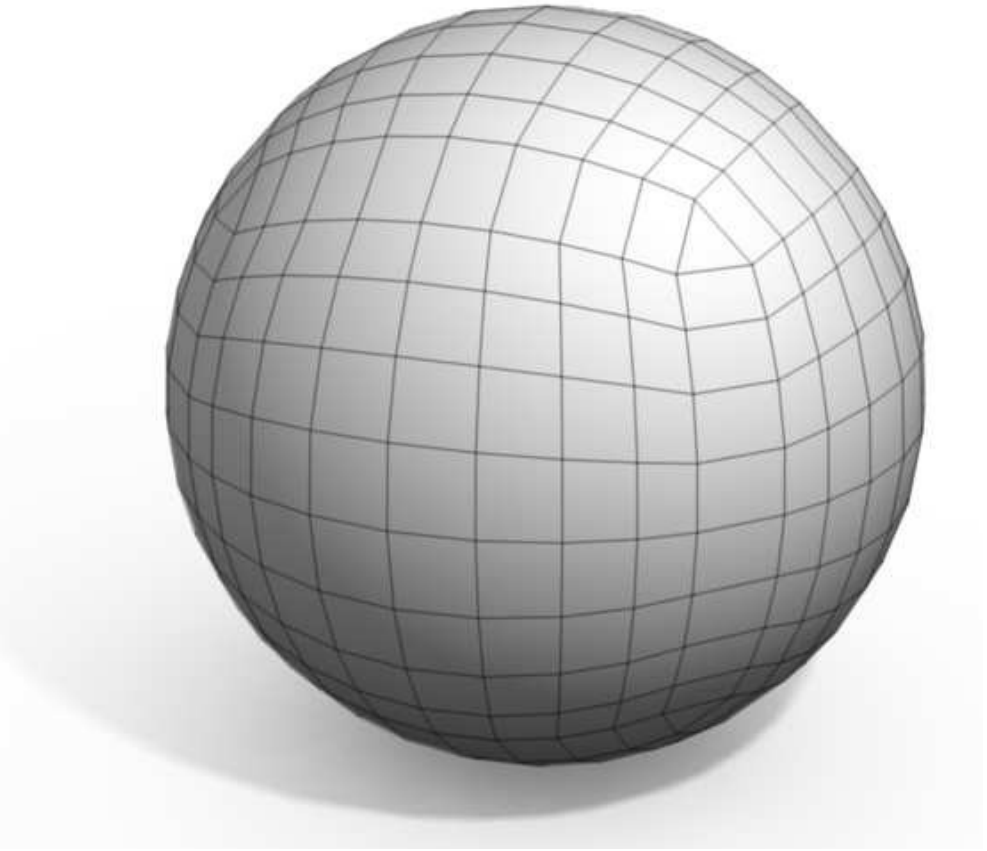




- Types of 3D mesh:
  - triangular, quadrilateral (4-sided), polygonal (n-sided)



triangular mesh

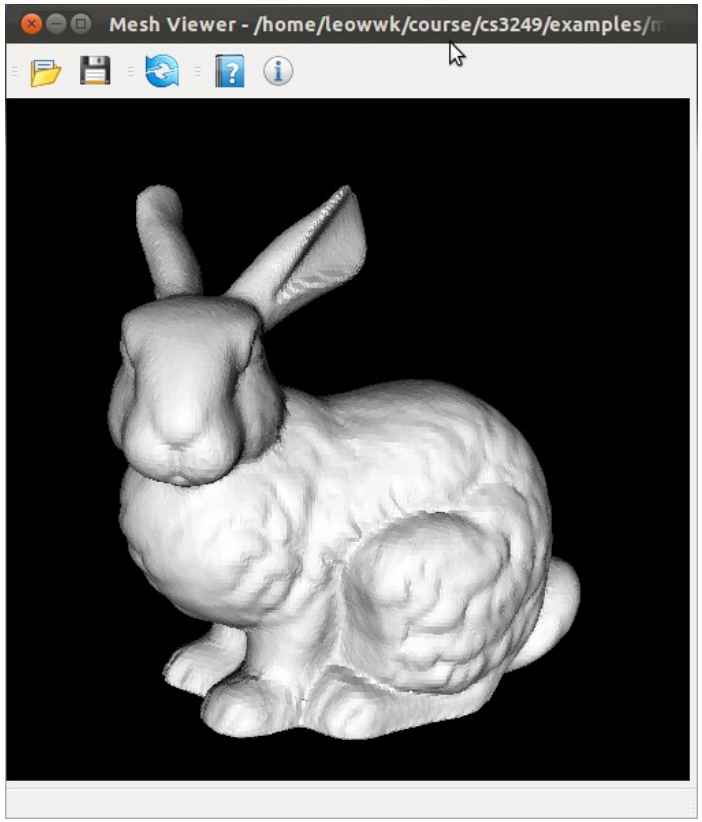


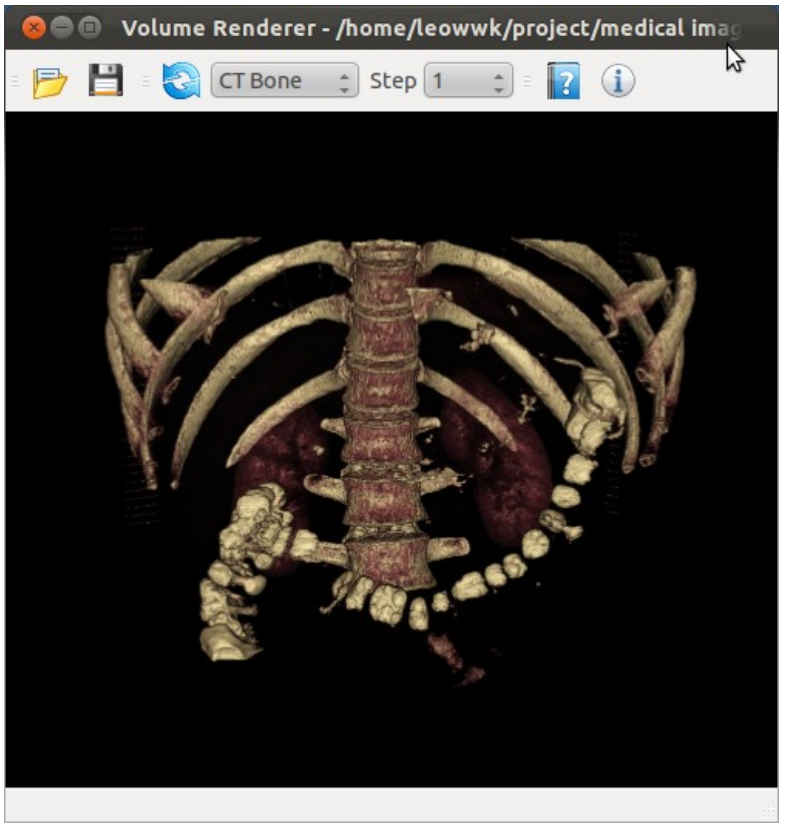
quadrilateral mesh

- To visualise 3D mesh
  - Need to support panning, rotation, zooming in/out.
- To visualise volume data
  - Need to support panning, rotation, zooming in/out.
  - Need to show interior of volume data.

VTK

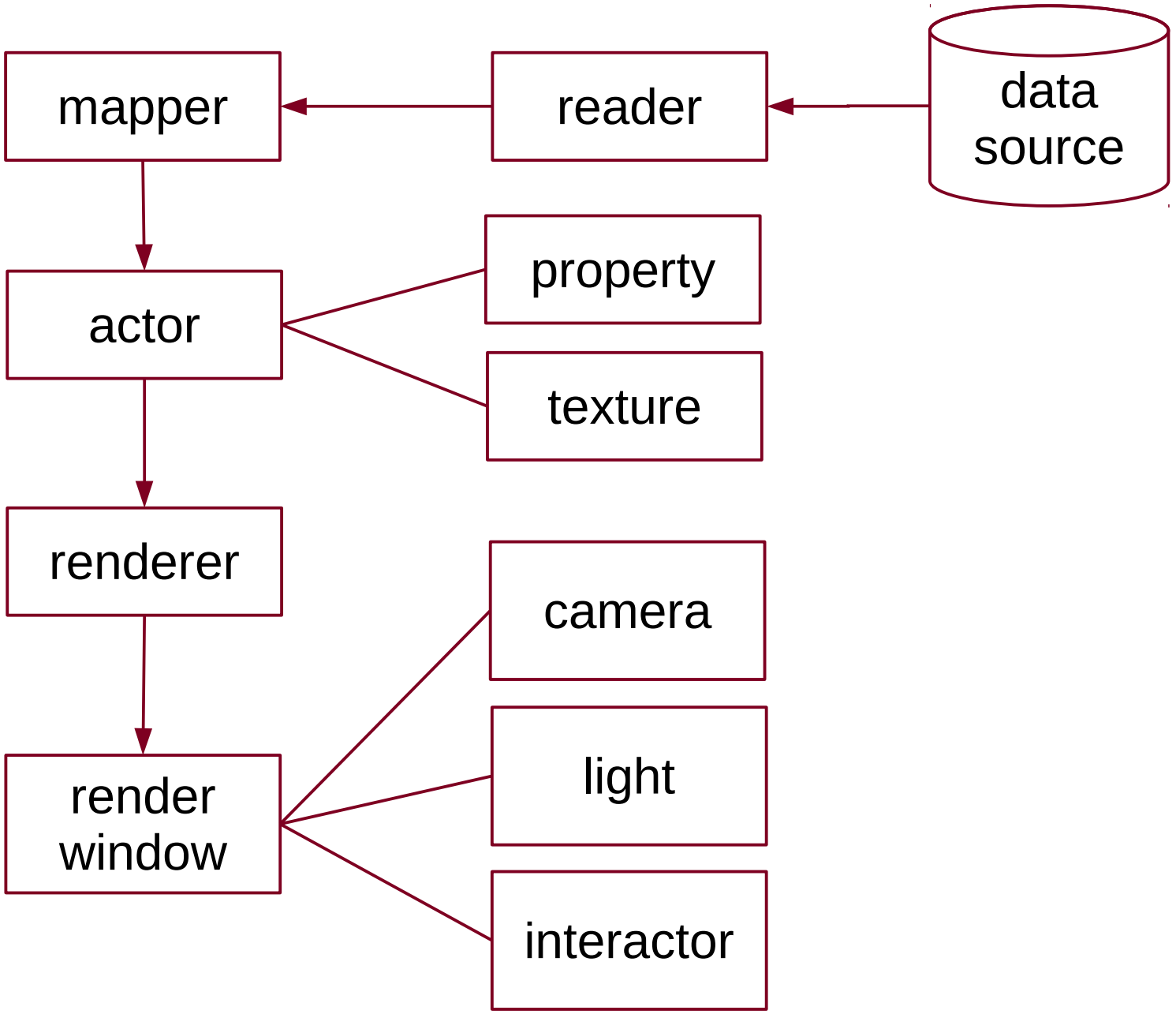
- The Visualization Toolkit
  - Open-source cross-platform toolkit for visualization of multi-dimensional data.





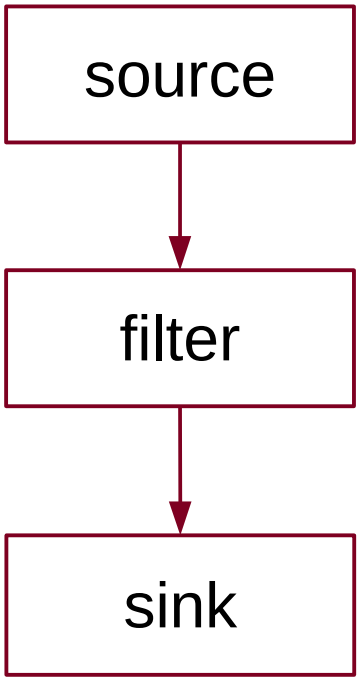
- Generic VTK application consists of
  - Data source: data file reader or predefined vtk data generator.
  - `vtkMapper`: maps data to graphics primitives.
  - `vtkActor`: represents object in a rendering scene.
    - `vtkProperty`: represents geometric and lighting properties.
    - `vtkTexture`: contains textures for 2D texture mapping.
  - `vtkCamera`: virtual camera through which data is viewed.
  - `vtkLight`: virtual light that illuminates the scene.
  - `vtkRenderer`: generates image from data through camera.
  - `vtkRenderWindow`: the window that renderer draws on.
  - `vtkRenderWindowInteractor`: supports user interactions.

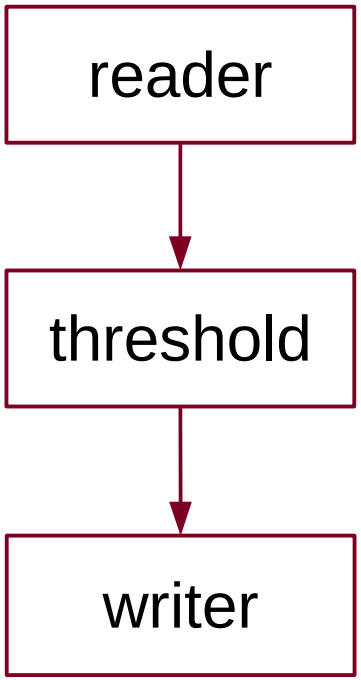
Generic Application

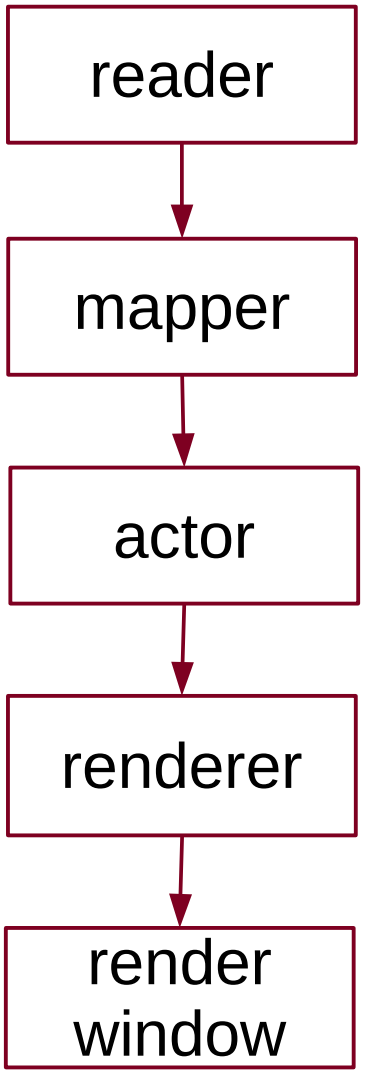


Data Flow Pipeline

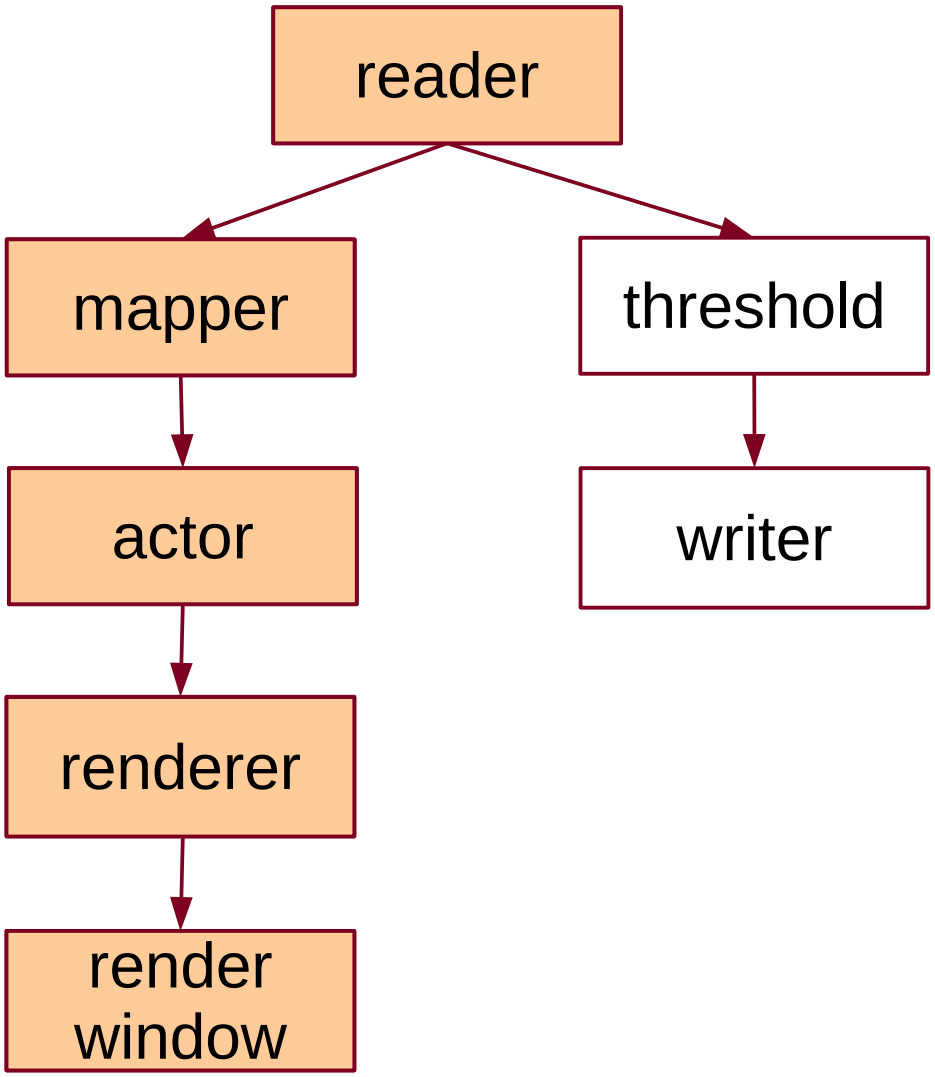
- VTK adopts data flow approach:
  - source has outputs only
  - filter has inputs and outputs
  - sink has inputs only







- VTK adopts lazy evaluation
  - Constructing pipeline doesn't execute it.
  - Call `update` or `render` of leaf node to start execution.
  - Recursively call parent nodes to execute.



VTK Viewers

- We illustrate 4 types of viewers
  - 2D image viewer
  - 3D mesh viewer
  - volume renderer
  - volume image viewer

2D Image Viewer

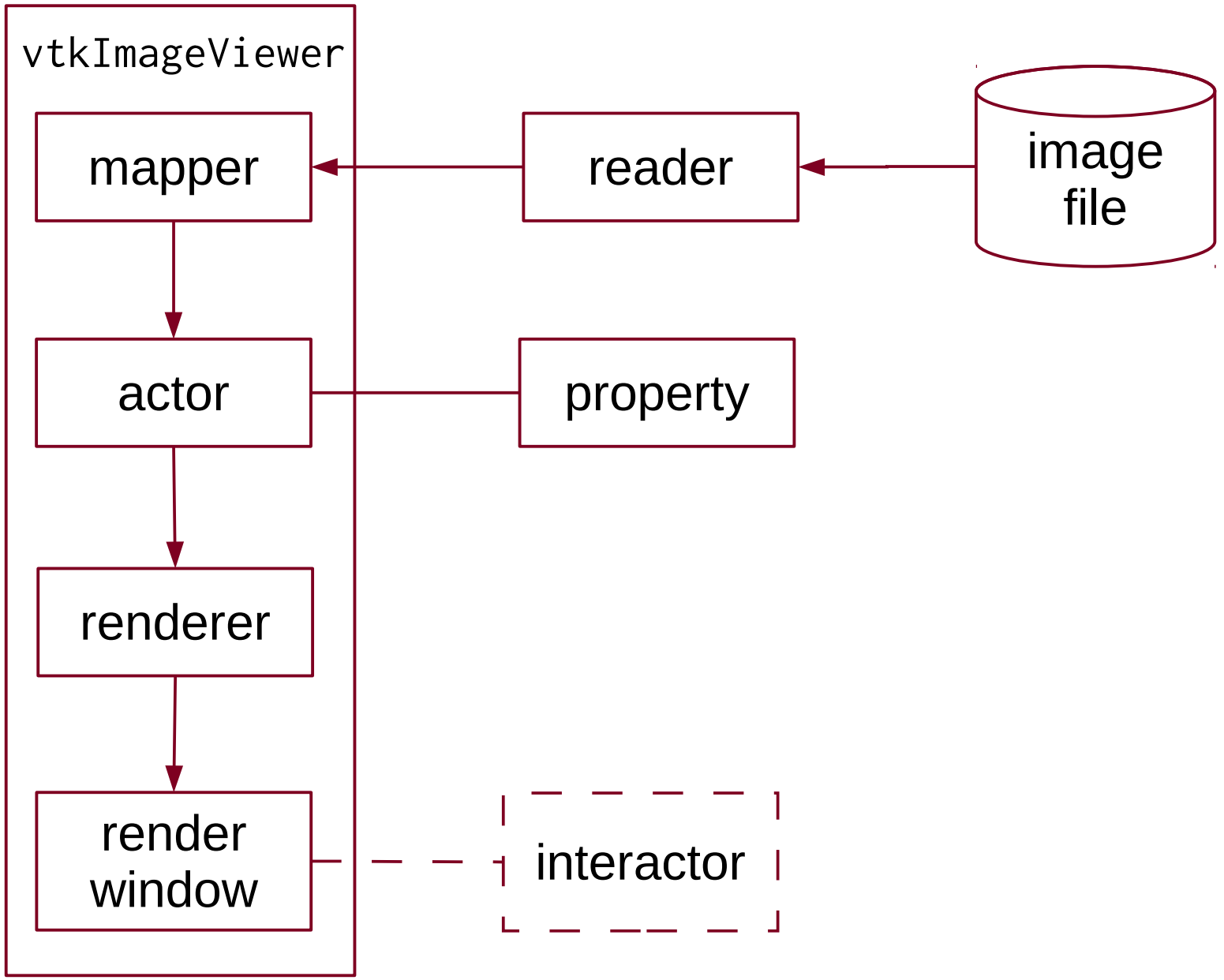
- VTK has two built-in image viewers
  - `vtkImageViewer`
    - for displaying 2D colour images
    - no zooming function
  - `vtkImageViewer2`
    - for displaying grey images, e.g., medical volume images
    - has zooming function
- First, let's try `vtkImageViewer`.

```
int main(int argc, char** argv)
{
    QApplication app(argc, argv);

    QVTKWidget *widget = new QVTKWidget;
    widget->resize(256, 256);

    vtkJPEGReader* reader = vtkJPEGReader::New();
    reader->SetFileName("butterfly.jpg");
    printSize(reader, 1);

    vtkImageViewer* viewer = vtkImageViewer::New();
    viewer->SetInputConnection(reader->GetOutputPort());
    viewer->SetColorLevel(128);
    viewer->SetColorWindow(256);
```

- `QVTKWidget`: a VTK widget that is also a `QWidget`.
  - Allow VTK to work inside Qt.

- `vtkImageViewer` packages the following together:
  - `vtkImageMapper`: mapper
  - `vtkActor2D`: actor
  - `vtkRenderer`: renderer
  - `vtkRenderWindow`: render window

- After connecting reader to `vtkImageViewer`...



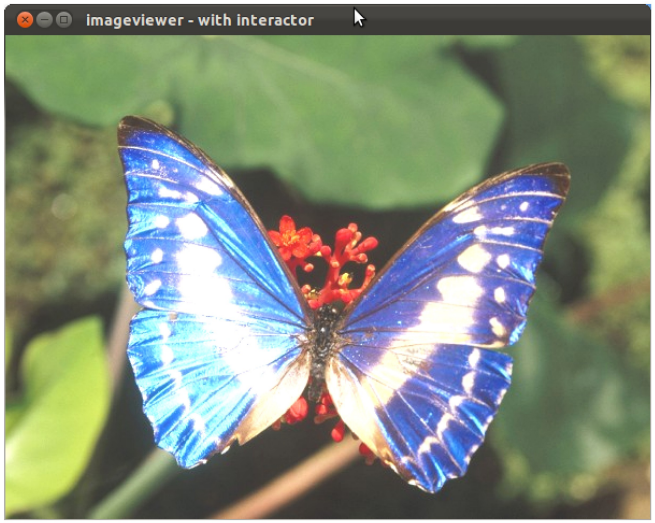
```
    vtkRenderWindow *renderWindow =
        viewer->GetRenderWindow();
    widget->SetRenderWindow(renderWindow);
    // viewer->SetupInteractor(
    //     renderWindow->GetInteractor());
    printSize(reader, 2);

    renderWindow->Render();
    printSize(reader, 3);

    int *size = imageSize(reader);
    widget->resize(size[0], size[1]);
    widget->show();
    app.exec();

    // Clean up
    viewer->Delete();  // VTK style
    reader->Delete();
    delete widget;     // Qt / C++ style
}
```

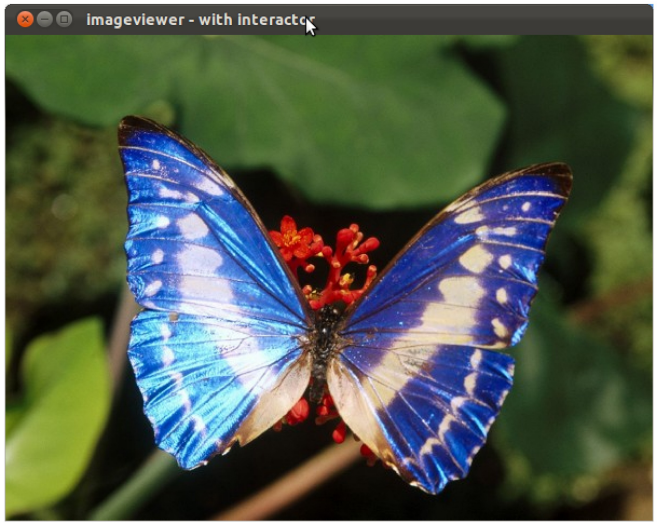
- Observe these outputs:

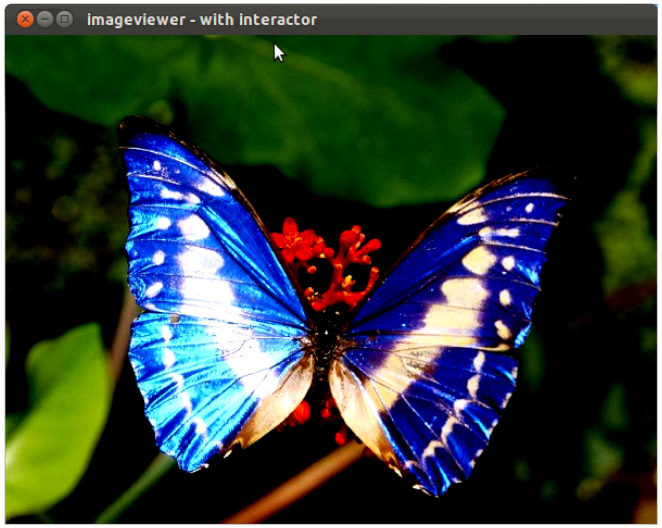
```
1: image size = 1, 1
2: image size = 1, 1
3: image size = 600, 450
```

- At 1 and 2:
  - Haven't executed pipeline.
  - Image size is set at default (1, 1).
- At 3:
  - Pipeline is executed.
  - Image is read and image size is the correct size.

- `SetupInteractor` sets up interactor
  - Uses render window's interactor if exists.
  - Otherwise, creates an appropriate interactor.
- Interactor allows user to change brightness and contrast by mouse motion.

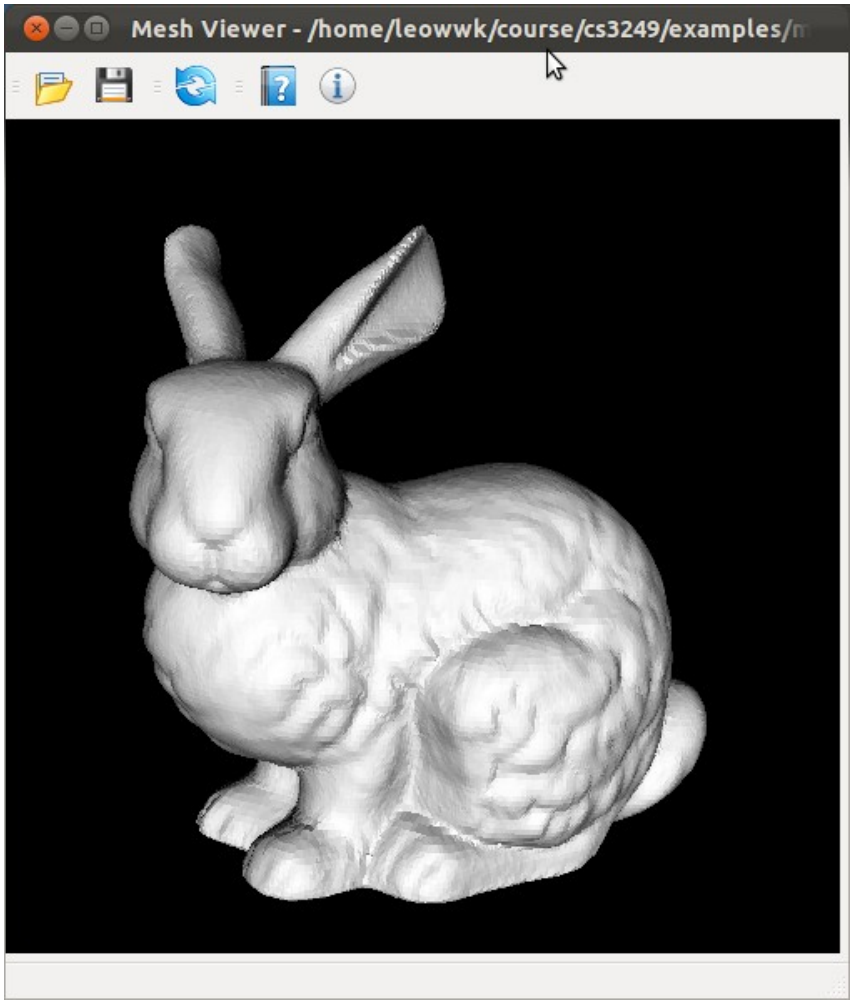






3D Mesh Viewer

- Visualise 3D surface mesh.



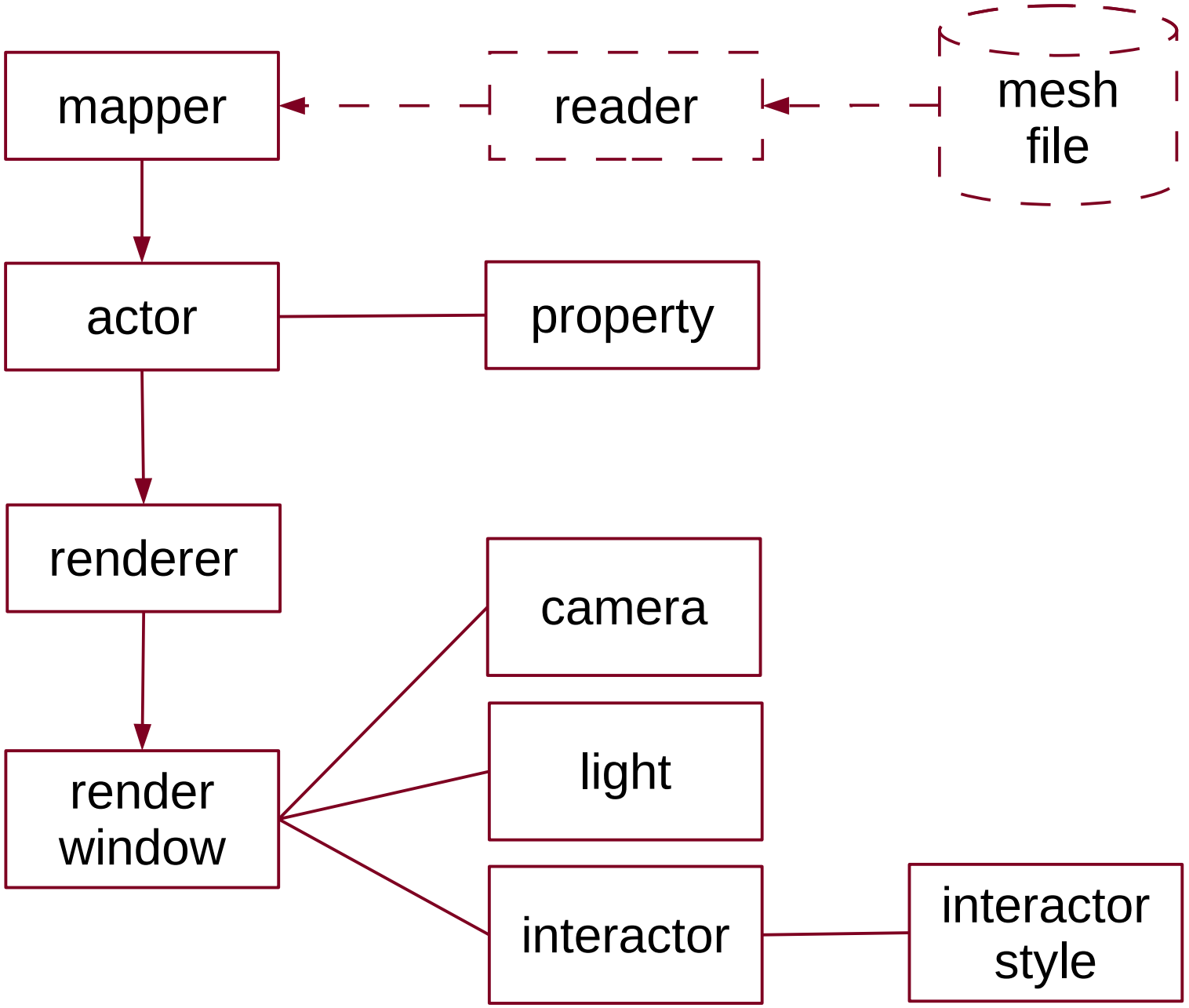
- Main VTK components for mesh viewer
  - reader: `vtkPolyDataAlgorithm`
  - mapper: `vtkPolyDataMapper`
  - actor: `vtkActor`
  - renderer: `vtkRenderer`
  - render window: `vtkRenderWindow`
  - interactor: `vtkRenderWindowInteractor`
  - interactor style: `vtkInteractorStyleTrackballCamera`

```
// MeshViewr.cpp

MeshViewer::MeshViewer()
{
    // Initialisation
    reader = NULL;

    // Create GUI
    createWidgets();
    createActions();
    createMenus();
    createToolBars();
    createStatusBar();
    initSize();
}
```

```
void MeshViewer::createWidgets()
{
    // Create vtk objects
    vtkWidget = new QVTKWidget(this);

    // Create mapper and actor
    mapper = vtkPolyDataMapper::New();
    actor = vtkActor::New();
    actor->SetMapper(mapper);

    // Create renderer
    renderer = vtkRenderer::New();
    renderer->AddActor(actor);
    renderer->SetBackground(0.0, 0.0, 0.0);

    // QVTKWidget has render window and interactor
    renderWindow = vtkWidget->GetRenderWindow();
    renderWindow->AddRenderer(renderer);
    interactor = renderWindow->GetInteractor();
```

```
    style = vtkInteractorStyleTrackballCamera::New();
    interactor->SetInteractorStyle(style);

    // Central widget
    setCentralWidget(vtkWidget);

    // Overall
    setWindowTitle("Mesh Viewer");
    setWindowIcon(QIcon(":/images/viewer.png"));
}
```

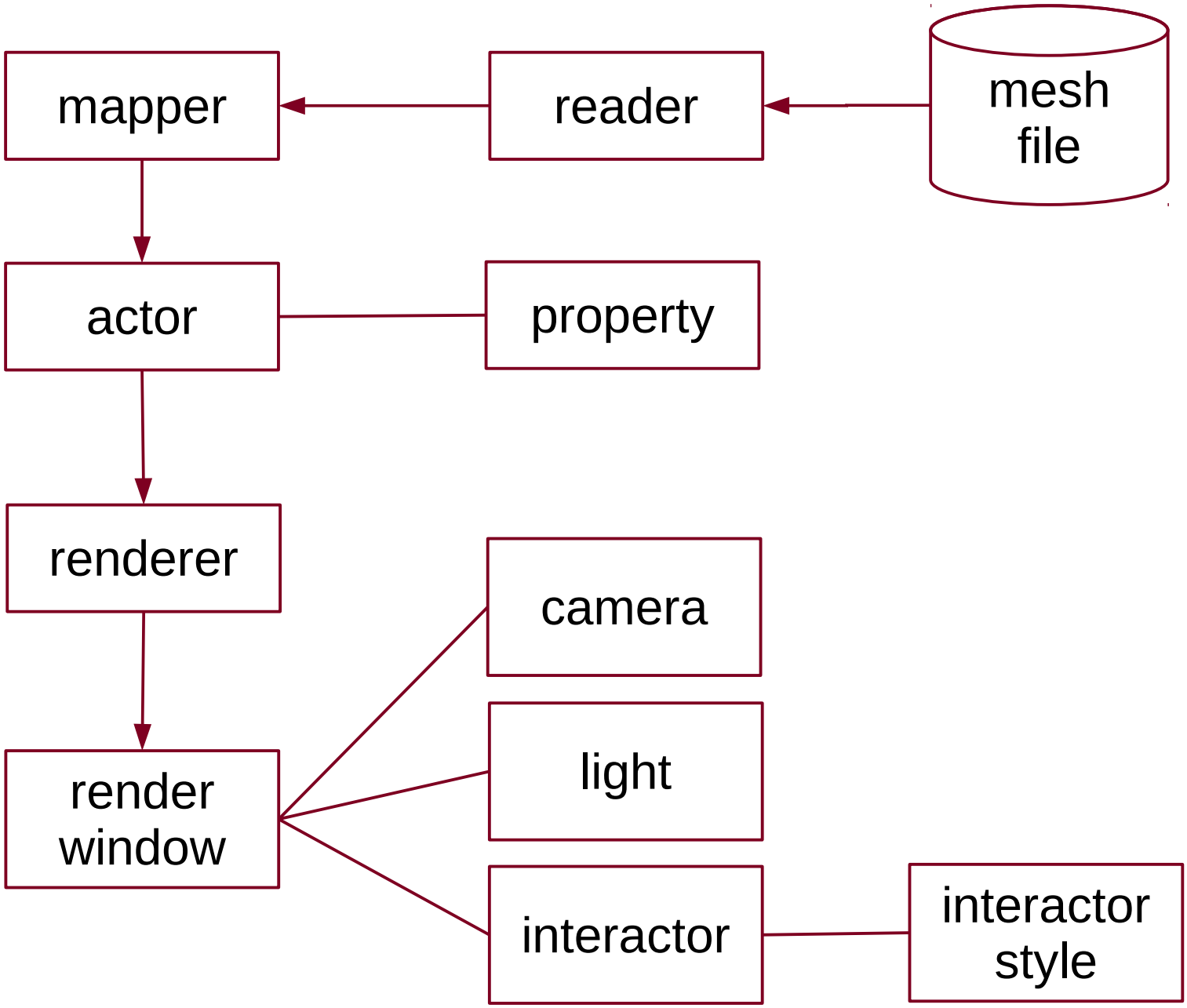
- After creating widgets...



```
void MeshViewer::loadMesh(const QString &fileName)
{
    QString suffix = QFileInfo(fileName).suffix();

    if (reader) {
        initCamera();
        reader->Delete();  // Remove current reader.
        reader = NULL;
    }

    if (suffix == "obj") {
        vtkOBJReader *objreader = vtkOBJReader::New();
        objreader->SetFileName(fileName.toAscii().data());
        reader = objreader;
    }
    else if (suffix == "ply") {
        vtkPLYReader *plyreader = vtkPLYReader::New();
        plyreader->SetFileName(fileName.toAscii().data());
        reader = plyreader;
    }
```

```
    else // This should not happen, but just in case.
    {
        cout << "Error in loadMesh: file type " <<
            suffix.toAscii().data() << " is unsupported.\n"
            << flush;
        return;
    }

    mapper->SetInputConnection(reader->GetOutputPort());
    renderer->ResetCamera();
    getCameraParameters();
    setWindowTitle("Mesh Viewer - " + fileName);
}
```

- After loading mesh from file...

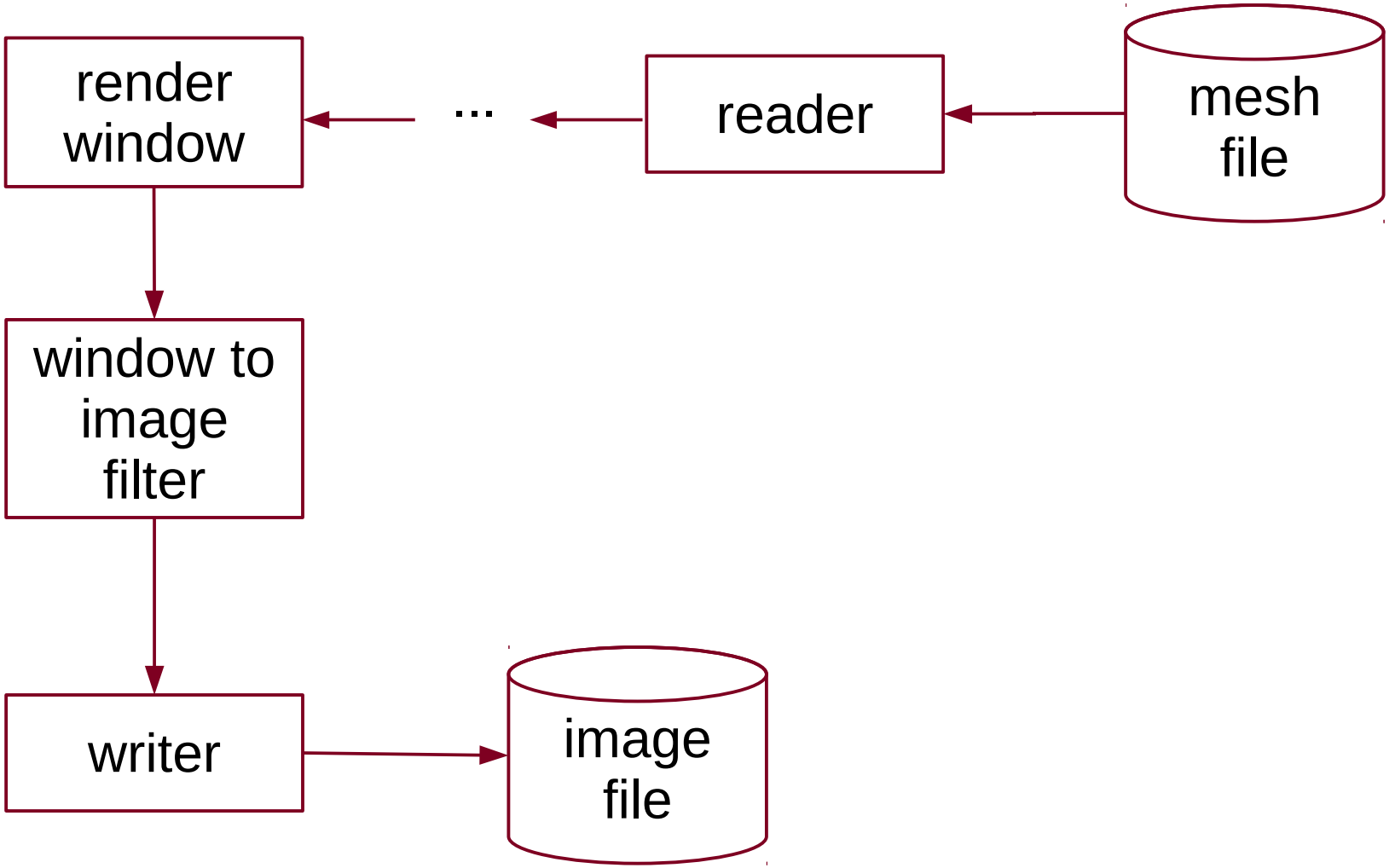


Save current view

```
bool MeshViewer::saveImage(const QString &fileName)
{
    vtkWindowToImageFilter *filter =
        vtkWindowToImageFilter::New();
    filter->SetInput(vtkWidget->GetRenderWindow());

    QString suffix = QFileInfo(fileName).suffix();
    vtkImageWriter *writer;

    if (suffix == "jpg")
        writer = vtkJPEGWriter::New();
    else if (suffix == "png")
        writer = vtkPNGWriter::New();
    else if (suffix == "tif")
        writer = vtkTIFFWriter::New();
```

```
    else // This should not happen, but just in case
    {
        cout << "Error in saveImage: File type " <<
            suffix.toAscii().data() << " is unsupported.\n"
            << flush;
        return false;
    }

    writer->SetInput(filter->GetOutput());
    writer->SetFileName(fileName.toAscii().data());
    writer->Write();
    writer->Delete();
    filter->Delete();
    return true;
}
```

- Output pipeline

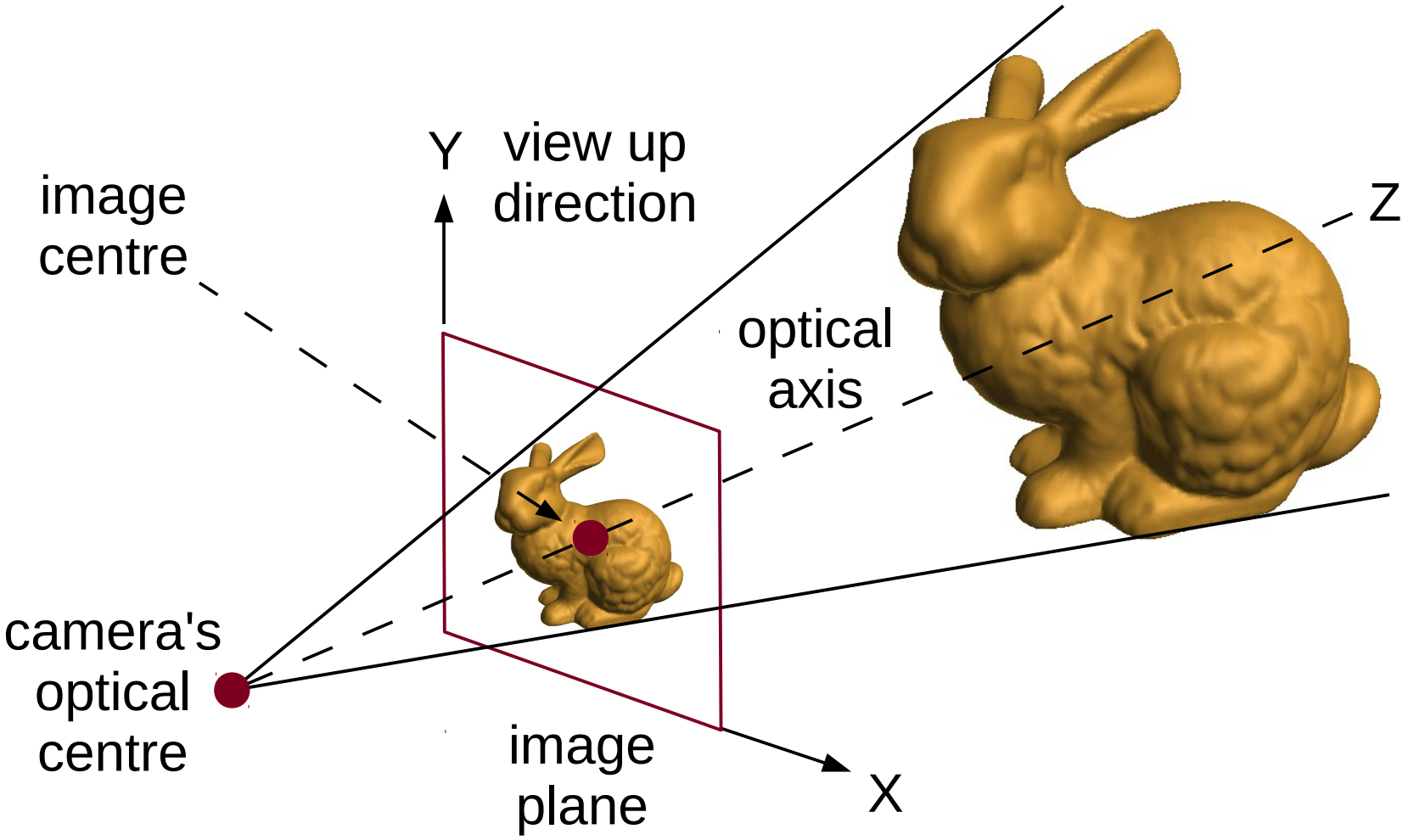


- Clean up
  - Delete VTK objects created.
  - No need to explicitly delete `QVTKWidget`.
    - It is deleted by its parent `QMainWindow`, along with other widgets.

```
MeshViewer::~MeshViewer()
{
    if (reader)
        reader->Delete();
    mapper->Delete();
    actor->Delete();
    renderer->Delete();
}
```

Image View

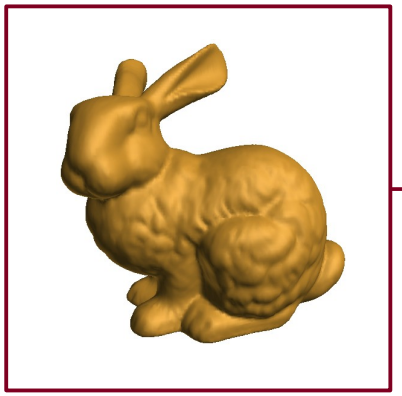
- Perspective imaging model



- Optical centre
  - The point where all rays converge.
  - VTK calls it camera position. Default: (0, 0, 1).
- Image centre
  - The point in image plane where the optical axis intersects.
  - VTK calls it focal point. Default: (0, 0, 0).
- Viewing direction
  - The direction from the camera position to the focal point.
- View's upward direction
  - Viewing direction + upward direction give camera orientation.
  - VTK calls upward direction view up. Default: (0, 1, 0).

- VTK uses 4 coordinate systems
  - Model coordinate system (3D)
    - For defining coordinates of points on an object, e.g., coordinates given in an object's mesh file.
  - World coordinate system (3D)
    - For defining positions of light, camera and actor.
  - View coordinate system (2D)
    - Coordinate system on the image plane.
  - Display coordinate system (2D)
    - Actual pixel coordinates on the display screen.

model coordinate system

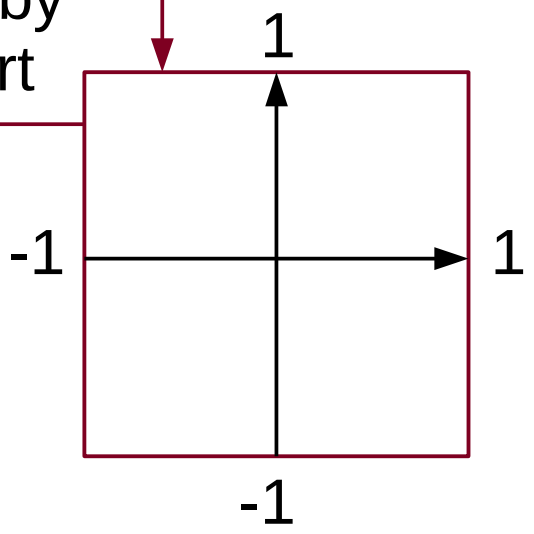


mapped by actor

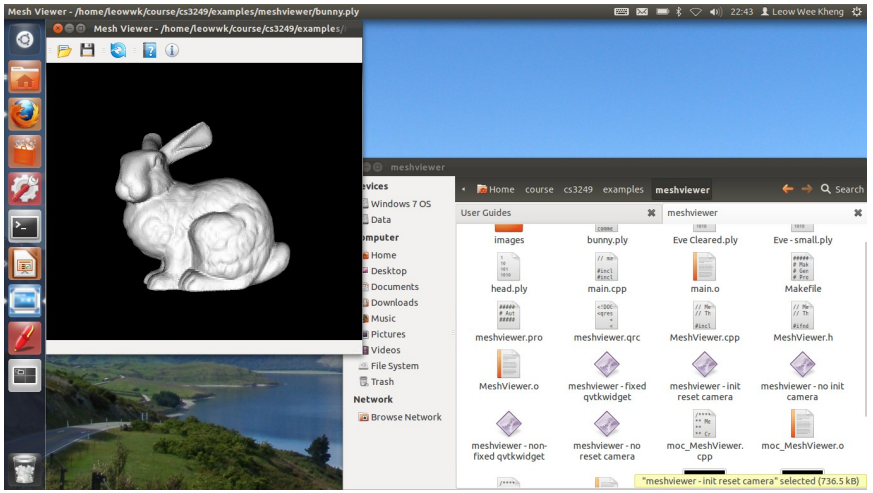
world coordinate system

light position
camera position
actor position

mapped by camera



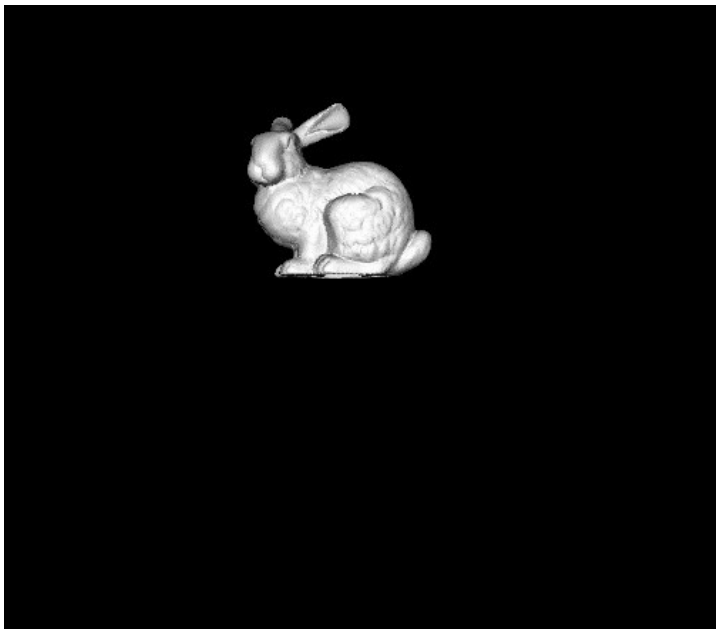
mapped by view port



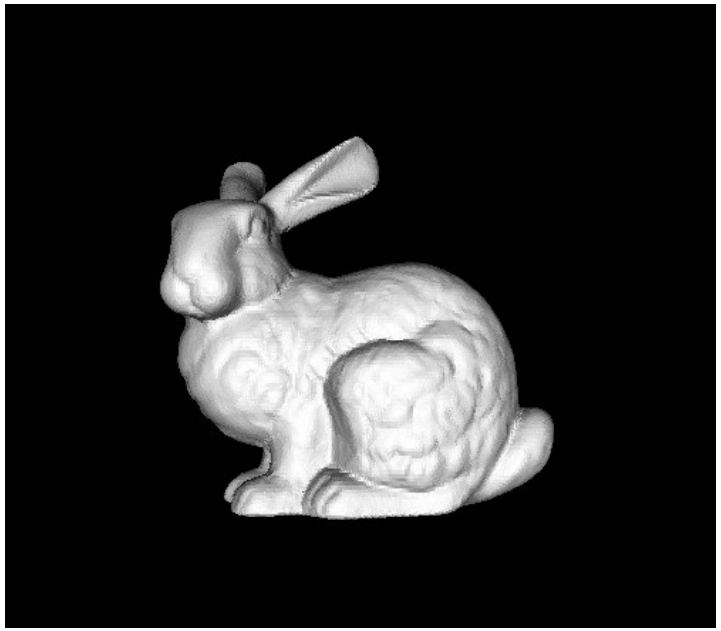
display coordinate system

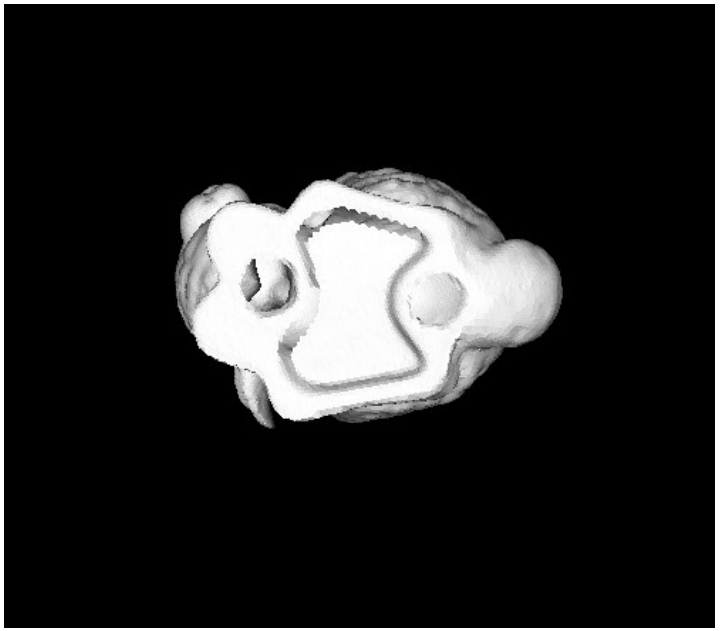
view coordinate system

- After loading object, have to set focal point and camera position for suitable viewing.
  - Otherwise, may end up like this:



- `vtkRenderer::ResetCamera`
  - Position camera to view all actors.
    - Place focal point at centre point of actors.
  - But doesn't set view up.
  - Loaded object is viewed from current viewing orientation.
    - Can cause inconsistency in initial viewing orientation.





- Solution: re-initialise camera parameters (and view up).
- Method 1:
  - After loading (first) object, keep initial camera parameters.
  - Before loading new object, reset camera parameters.
  - After loading new object, invoke ResetCamera.
- Method 2:
  - Remove renderer.
  - Re-create renderer,
    which re-creates camera with default view up.
  - After loading object, invoke ResetCamera.

```
void MeshViewer::loadMesh(const QString &fileName)
{
    if (reader)
    {
        initCamera();
        reader->Delete();
        reader = NULL;
    }
    ...
    mapper->SetInputConnection(reader->GetOutputPort());
    renderer->ResetCamera();
    getCameraParameters();
    setWindowTitle("Mesh Viewer - " + fileName);
}


void MeshViewer::initCamera()
{
    // Method 1
    resetCameraParameters();
}
```

```
void MeshViewer::getCameraParameters()
{
    vtkCamera *camera = renderer->GetActiveCamera();
    camera->GetFocalPoint(cameraFocalPoint);
    camera->GetPosition(cameraPosition);
    camera->GetViewUp(cameraViewUp);
}


void MeshViewer::resetCameraParameters()
{
    vtkCamera *camera = renderer->GetActiveCamera();
    camera->SetFocalPoint(cameraFocalPoint);
    camera->SetPosition(cameraPosition);
    camera->SetViewUp(cameraViewUp);
}
```

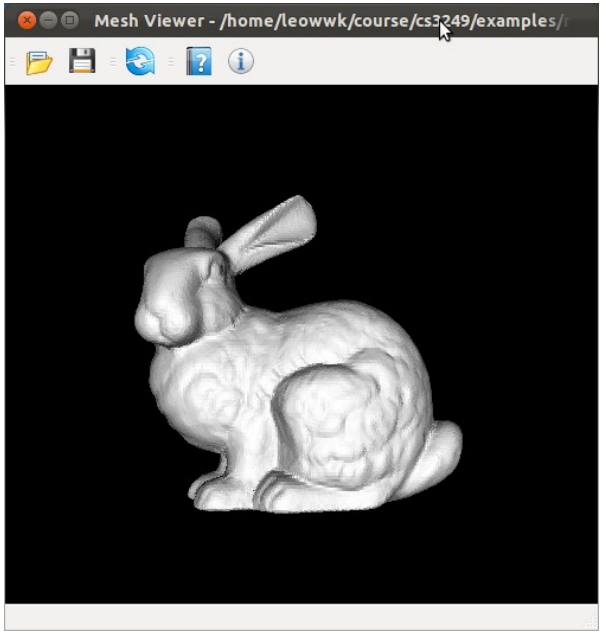
```
void MeshViewer::initCamera()
{
    // Method 2
    if (renderer)
    {
        renderWindow->RemoveRenderer(renderer);
        renderer->Delete();
        renderer = NULL;
    }

    renderer = vtkRenderer::New();
    renderer->AddActor(actor);
    renderer->SetBackground(0.0, 0.0, 0.0);
    renderWindow->AddRenderer(renderer);
}
```

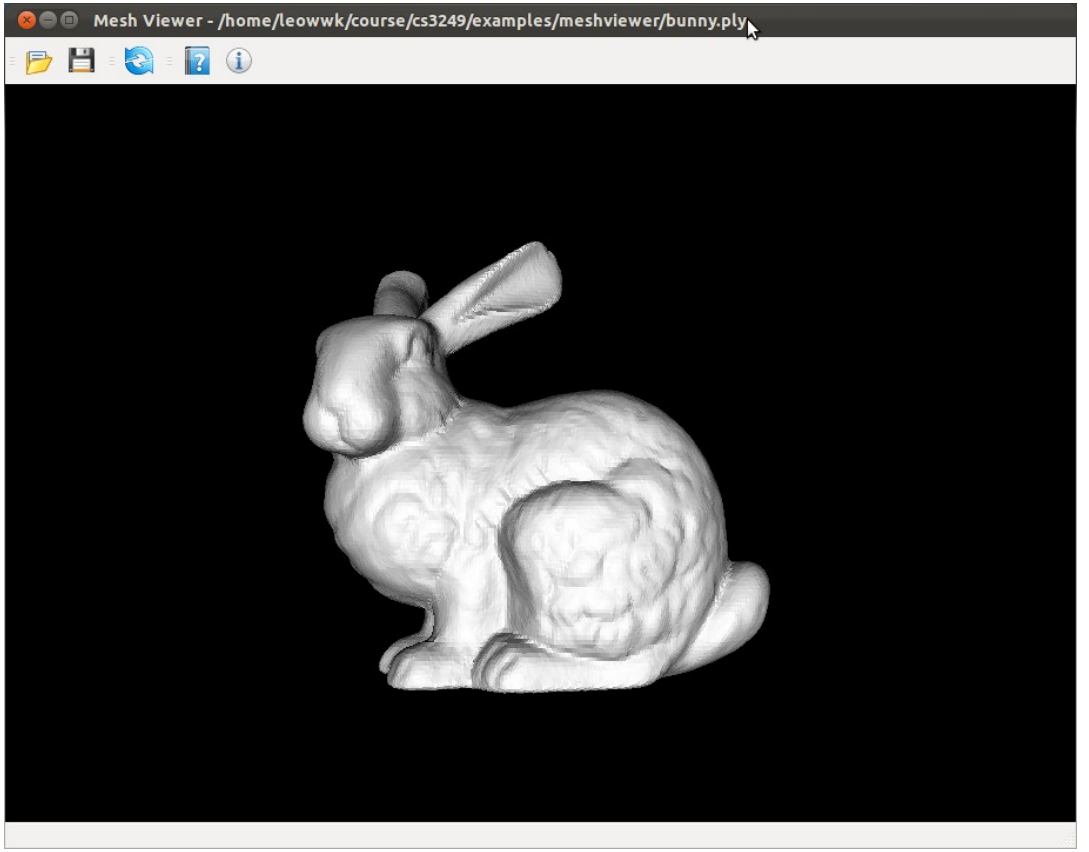
Window Size

- Window size can be initialised as follows:

```
void MeshViewer::initSize()
{
    winWidth = 500;
    winHeight = 500;
    setMinimumSize(winWidth, winHeight);

    QRect rect = geometry();  // Current window's geometry.
    setGeometry(
        rect.left(), rect.top(),  // Window's position
        winWidth, winHeight);     // Window size

    vtkWidget->updateGeometry();
}
```

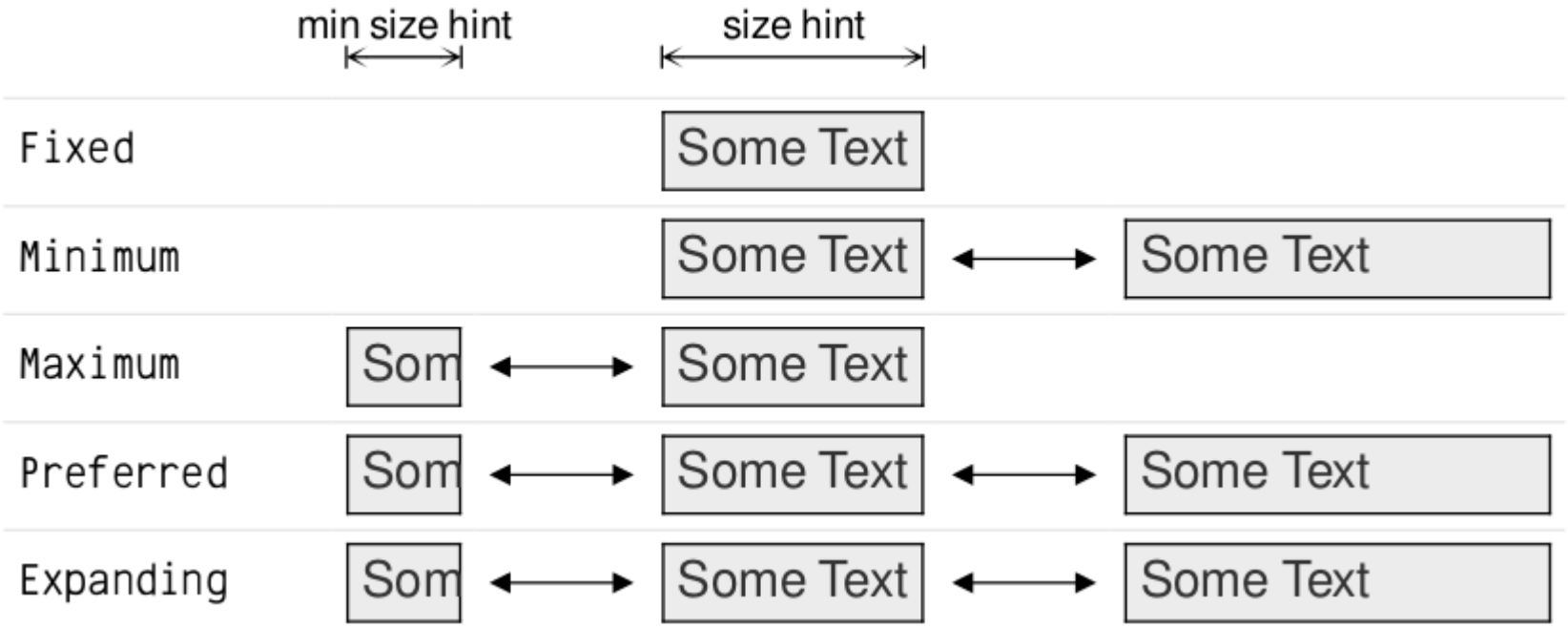
- `QVTKWidget` is central widget of `QMainWindow`.
- `QVTKWidget` auto resizes with `QMainWindow`.
- Object's image auto resizes with `QVTKWidget`
  - No change in camera parameters.
  - Only size of image plane is changed.



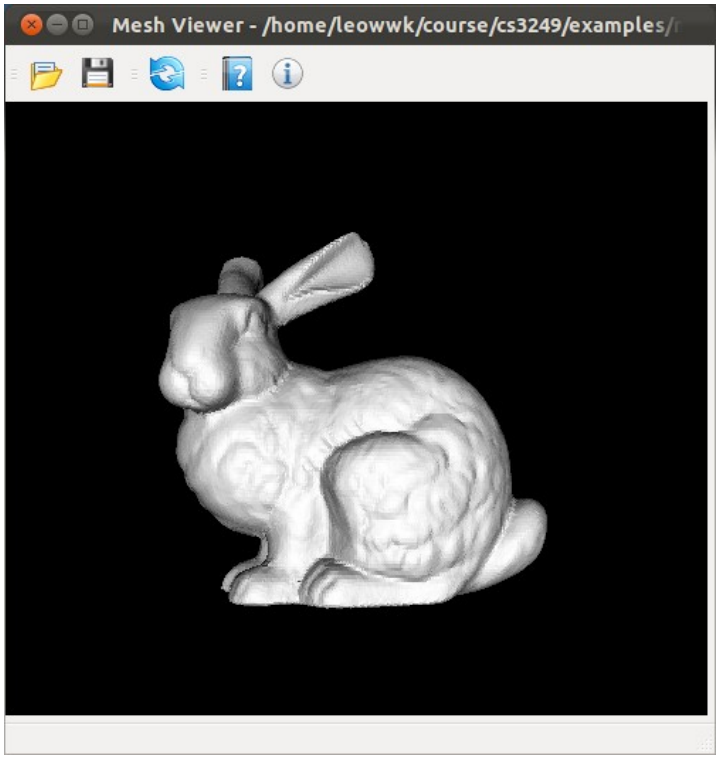


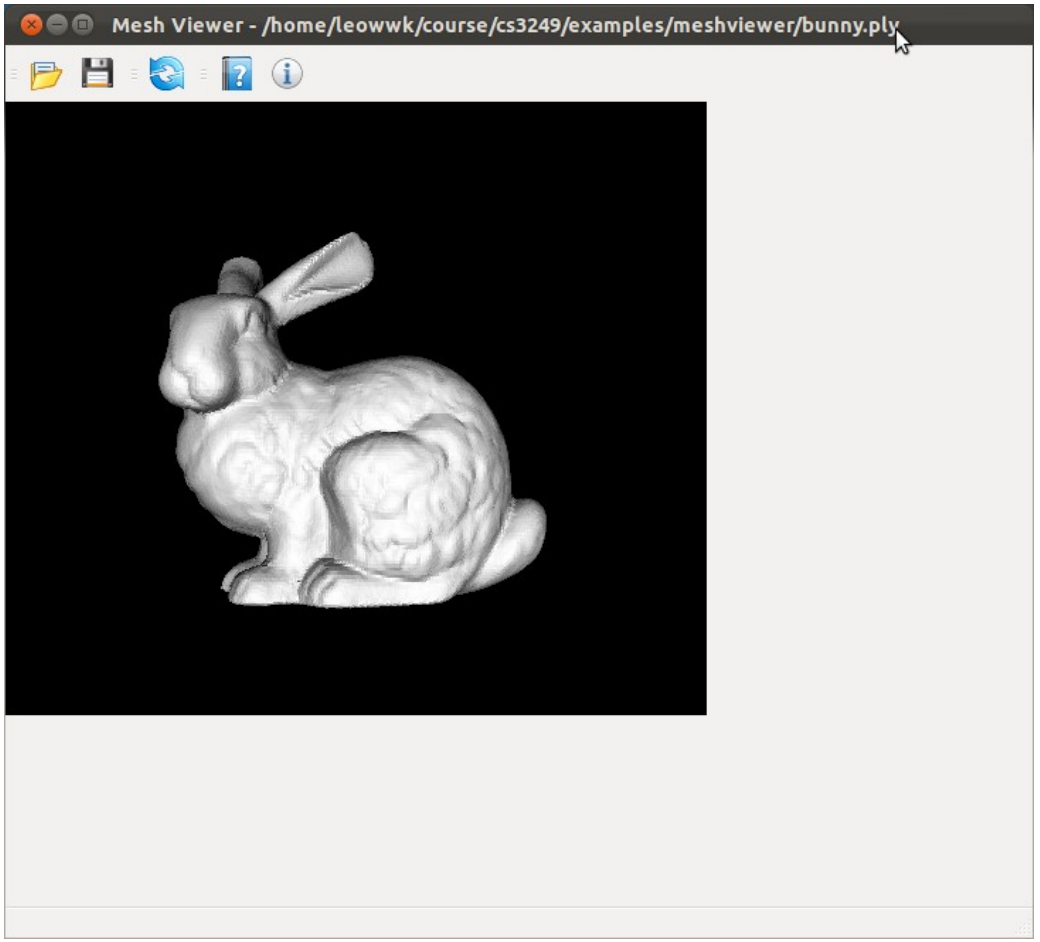
- Most Qt widgets have 4 size attributes
  - minimum size: default (0, 0)
  - maximum size: default (16777215, 16777215)
  - size hint: recommended size
  - minimum size hint: recommended minimum size
- Widget size can be changed by
  - `resize()` : set to specified size.
  - `setMinimumSize()` : set minimum size, overrides min size hint.
  - `setMaximumSize()` : set maximum size, overrides default.
  - `setFixedSize()` : set to fixed size
  - `adjustSize()` : adjust size to fit content.

- Qt Layout manager adjusts sizes of widgets in it.
- Widget's size policy indicates willingness to change:
  - `Fixed`: fixed to size hint
  - `Minimum`: size hint is minimum size
  - `Maximum`: size hint is maximum size
  - `Preferred`: size hint is ok, can be smaller or larger. Default.
  - `Expanding`: size hint is ok, but larger is better.



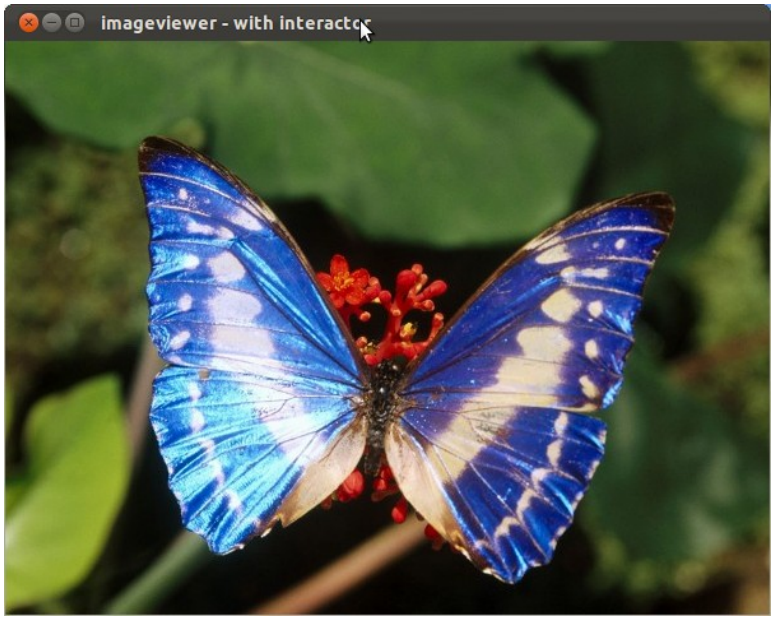
Fixed-size `QVTKWidget`

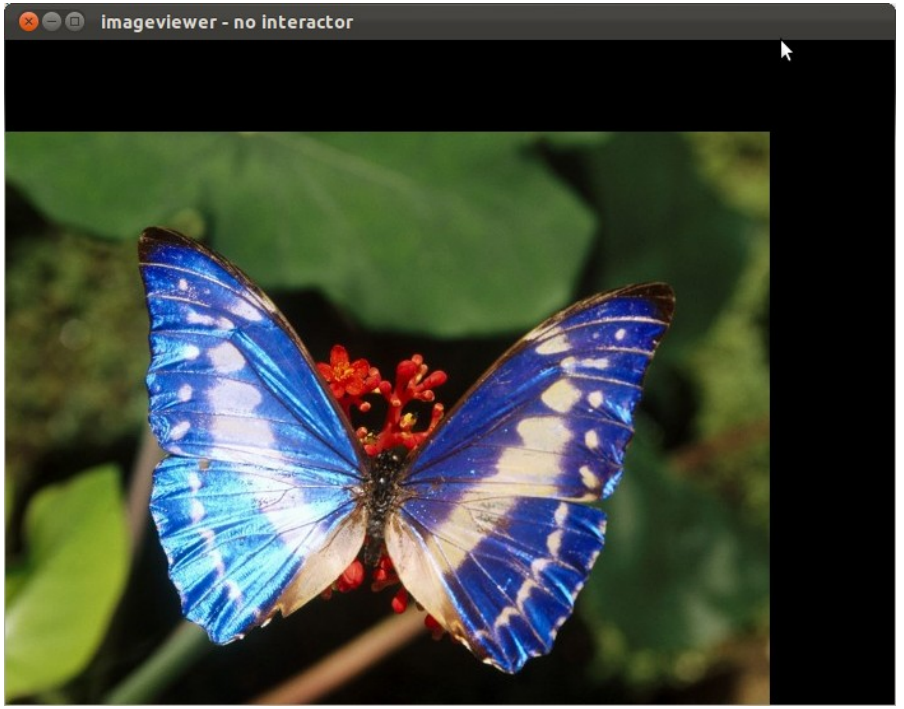




```
void MeshViewer::initSize()
{
    winWidth = 500;
    winHeight = 500;
    setMinimumSize(winWidth, winHeight);

    QRect rect = geometry();
    setGeometry(rect.left(), rect.top(),
        winWidth, winHeight);

    int winWMargin = 5;
    int winHMargin = 67;
    vtkWidget->setFixedSize(winWidth – winWMargin,
        winHeight - winHMargin);
    vtkWidget->setSizePolicy(QSizePolicy::Minimum,
        QSizePolicy::Minimum);

    vtkWidget->updateGeometry();
}
```

Fixed-size `vtkImageViewer`





Summary

- VTK adopts data flow pipeline and lazy evaluation.
- Main components
  - reader, mapper, actor, renderer, render window.
- Use `QVTKWidget` to incorporate VTK into Qt.
- Keep VTK camera parameters to reset view.
- Use `QSizePolicy` to set resize policy.

Further Reading

- VTK pipeline: [Schr2006] chap. 4.
- VTK components: [Schr2006] p. 58–60.
- `QSizePolicy`: [Blan2008] chap. 6, Qt Assistant.
- Complete source codes of mesh viewer: Lab 2.

References

- J. Blanchette and M. Summerfield, *C++ GUI Programming with Qt 4*, 2nd ed., Prentice Hall, 2008.
- W. Schroeder, K. Martin and B. Lorensen, *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th Ed., Kitware Inc., 2006.