

Rensselaer

**ESCE 4960: Open Source Software Practice
Lecture 7: VTK Overview
September 20, 2007**

Dr. Will Schroeder, Kitware

Background: Visualization

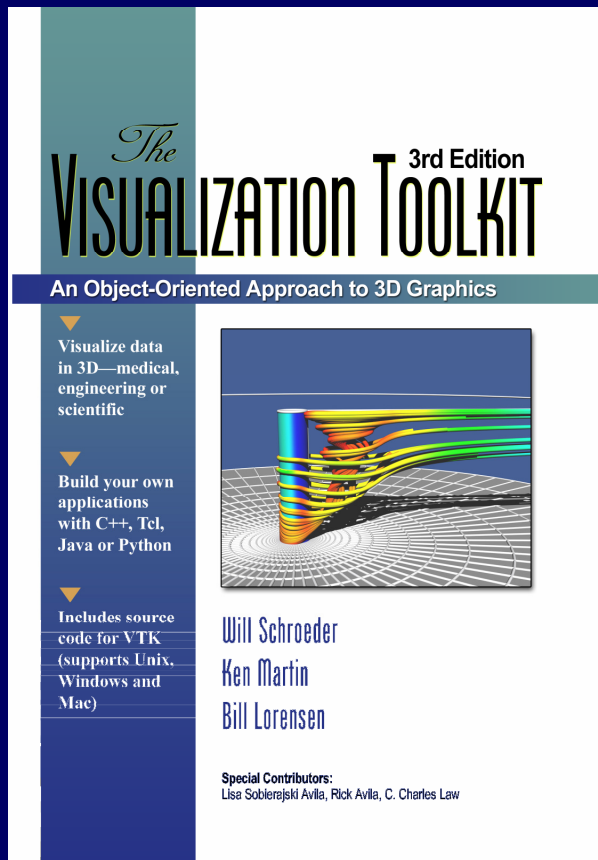
- Definition
 - Map data or information into images or other sensory input (touch, sound, smell, taste)
 - Engages human perception system
 - Simple, effective powerful
 - Complex data
 - Voluminous data

Visualization

- Related disciplines
 - 3D Graphics
 - Image processing
 - Modeling
 - Computational geometry
 - Numerical methods
 - Scientific and parallel computing
 - GUI and Computer/Human Interaction techniques
 - Perception / Human factors

Origins of VTK: Textbook

Now in Third Edition



The Visualization Toolkit
An Object-Oriented Approach To 3D Graphics
Will Schroeder, Ken Martin, Bill Lorensen
ISBN 1-930934-07-6
Kitware, Inc. Publishers

***Work on first edition began in
December 1993***

What Is VTK?

A visualization toolkit

- Designed and implemented using object-oriented principles
- C++ class library (750,000 LOC)
- Automated Java, TCL, Python language bindings
- Portable across Unix, Windows XP, MacOSX
- Supports 3D/2D graphics, visualization, image processing, volume rendering

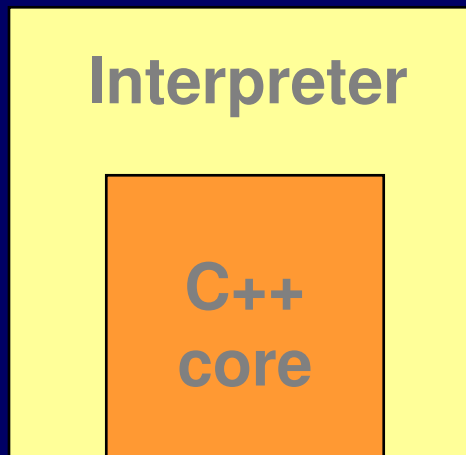
Building Applications in VTK

- Applications can be created using
 - C++ (compiled language)
 - Tcl (interpreted language)
 - Java (interpreted language)
 - Python (interpreted language)
- Interpreted Languages
 - Have GUI support
 - Easy to prototype with
 - Slower than compiled C++

VTK Architecture

Hybrid approach

- Compiled C++ core (faster algorithms)
- Interpreted applications (rapid development)
(Java, Tcl, Python)



*Interpreted layer
generated
automatically
by VTK wrapping
process*

How to Get The VTK Software

- VTK5.0 CD (Release version – most stable)
- Anonymous CVS (Development version – caution)
 - CVS is a source code control system
 - The command:

```
cvs -d :pserver:anonymous@public.kitware.com:/cvsroot/VTK  
co VTK
```

is used to checkout the source code (password “vtk”)
 - Release branch:

```
cvs -d [...] co -r VTK-5-0 VTK
```

VTK Directory Structure

./VTK

- ./VTK/Common** – *core VTK classes*
- ./VTK/Filtering** – *classes used to manage pipeline dataflow*
- ./VTK/Rendering** – *rendering, picking, image viewing, and interaction*
- ./VTK/VolumeRendering** – *volume rendering techniques*
- ./VTK/Graphics** – *3D geometry processing*
- ./VTK/GenericFiltering** – *non-linear 3D geometry processing*
- ./VTK/Imaging** – *imaging pipeline*
- ./VTK/Hybrid** – *classes requiring both Graphics and imaging functionality*
- ./VTK/Widgets** – *sophisticated interaction*
- ./VTK/IO** – *VTK input and output*
- ./VTK/Parallel** – *parallel processing (controllers and communicators)*
- ./VTK/Wrapping** – *support for Tcl, Python, and Java wrapping*
- ./VTK/Examples** – *extensive, well-documented examples*

VTK Documentation

- *VTK User's Guide*
- *The Visualization Toolkit* textbook
- On the VTK5.0 CD (Doxygen):
`cdrom:/vtkhtml/html/index.html`
- On the Web (Doxygen, current state):
`http://www.vtk.org/doc/release/5.0/html/`
- Embedded documentation in `.h` header files
- Search `VTK/Examples`, `VTK/*/Testing/Tcl` and `VTK/*/Testing/Cxx/` directories for usage

Doxygen

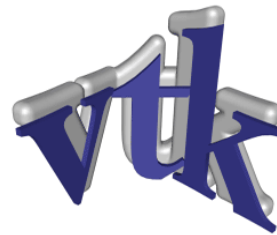
Alphabetical
listing of
classes

Method
names to
classes

Classes to
examples;
events to
classes

[Main Page](#) | [Class Hierarchy](#) | [Alphabetical List](#) | [Class List](#) | [Directories](#) | [File List](#) | [Class Members](#) | [File Members](#) | [Related Pages](#)

VTK 5.0.2 Documentation



Revision
1.2196

Date
2005/09/01 09:01:29

Useful links:

- VTK Home: <http://www.vtk.org>
- VTK Users Mailing-list: <http://public.kitware.com/mailman/listinfo/vtkusers>
- VTK Developer Mailing List: <http://www.vtk.org/mailman/listinfo/vtk-developers>
- VTK FAQ: http://www.vtk.org/Wiki/VTK_FAQ
- VTK Wiki: <http://www.vtk.org/Wiki/>
- VTK Search: <http://www.kitware.com/search.html>
- VTK Dashboard: <http://www.vtk.org/Testing/Dashboard/MostRecentResults-Nightly/Dashboard.html>
- VTK-Doxygen scripts (Sebastien Barre): <http://www.barre.nom.fr/vtk/doc/README>
- Kitware Home: <http://www.kitware.com>
- Sebastien's VTK Links: <http://www.barre.nom.fr/vtk/links.html>
- Other Links: <http://www.vtk.org/links.php>

Important File: `vtkSetGet.h`

- System-wide macro definitions
 - `vtkTypeRevisionMacro` – versioning information
 - `GetClassName()`
 - `IsTypeOf()`
 - `IsA()`
 - `vtkSet / vtkGet` Macros
 - `vtkDebugMacro / vtkWarningMacro / vtkErrorMacro`
 - `vtkStandardNewMacro()` – defines `New()` method
 - `vtkCxxRevisionMacro()` – versioning information for `.cxx` file

Example Set/Get Macros

- `vtkSetMacro(var,type) → void SetVar(type _arg);`
 - E.g., `SetDebug()`
- `vtkGetMacro(var,type) → type GetVar()`
 - E.g., `GetDebug()`
- `vtkBooleanMacro(var,type) →`
`void VarOn(), void VarOff()`
 - `DebugOn(), DebugOff()`
- `vtkSetVectorMacro(var,type,count) →`
`void SetVar(type data[count])`
 - `SetPosition(x[3])`

Exercise 1 : Preliminaries

- CMake \geq 2.4 is required for all exercises
 - Run on exercise directory
 - Will produce workspaces/makefiles as appropriate
 - Then run “make” or MSVC/nmake
- See <http://www.cmake.org> for more information

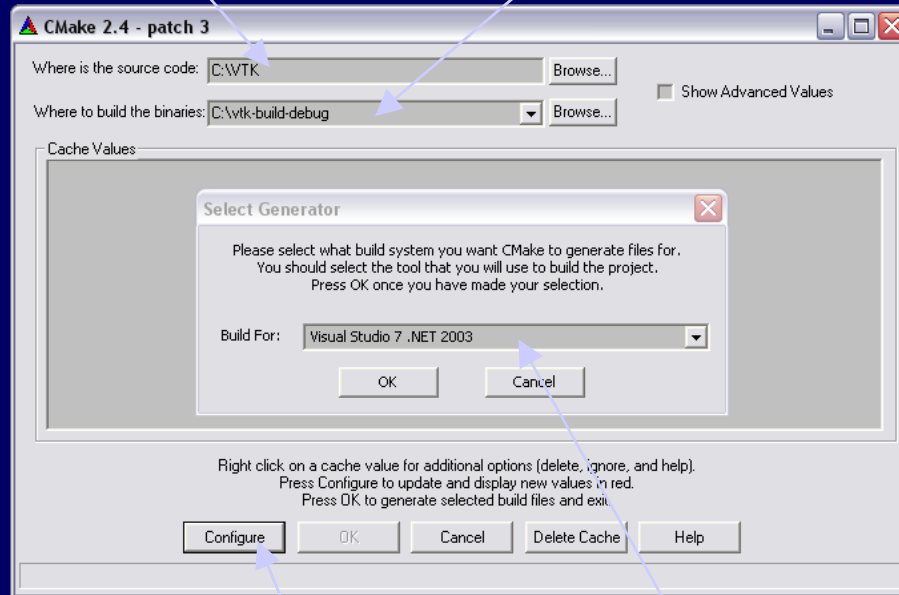
Running CMake

Source code location

Object code location

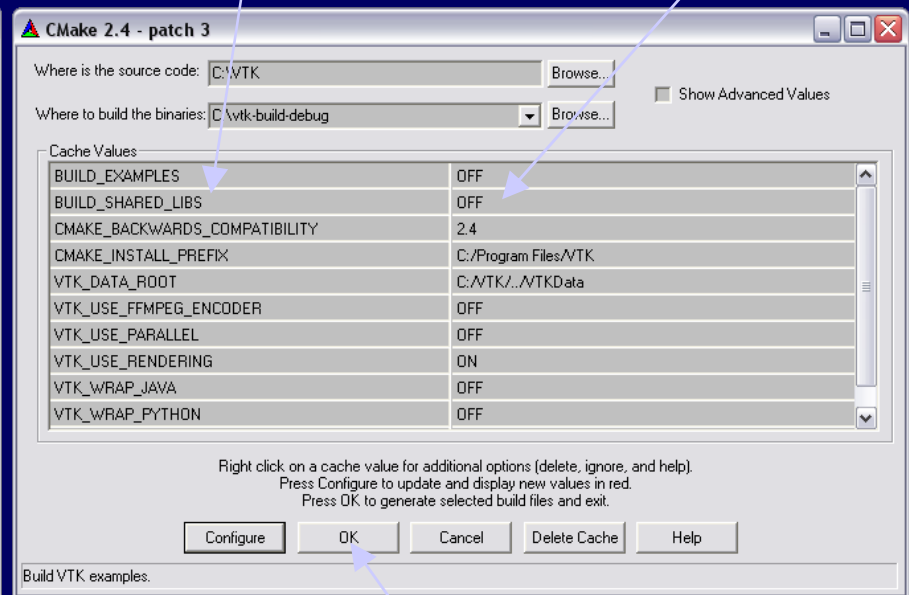
Build parameters

Build settings



Configure build

Compiler to use



Produce build files

Exercise 1

- Introduction
 - Run CMake on `exercise1/CMakeLists.txt`
 - Start Microsoft Visual C++ (or other compiler)
 - Load the `exercise1` workspace (or makefile)
 - Compile & run the program (keypress “e” exits)
 - Change the resolution of the cone
 - Look for `SetResolution()` method in Doxygen pages
 - Recompile and rerun

VTK Coding Standards

Coding standards result in code easier to understand and navigate

- One public class per file. Some classes have helper classes that they use, but these are not accessible to the user.
- Class names and file names are the same
- Every class, macro, etc. starts with either vtk or VTK. Classes should start with vtk macros or constants can start with either.

VTK Coding Standards

- Only alphanumeric characters are used in names, [a-zA-Z0-9]. Names like `Extract_Surface` are not welcome.
- Capitalization is used to indicate words within a name. For example `ExtractVectorTopology` could be an instance variable. If it were a class it would be called `vtkExtractVectorTopology`.
- Capitalize the first letter of a name (excluding any preceding `vtk`).

VTK Coding Standards

- For local variables almost anything goes. we suggest using the same convention as instance variables except start their names with a lower case letter.
- Spell out a name and do not use abbreviations. This leads to longer names but it makes using the software easier because you know that the `SetRasterFontRange` method will always be called that, not `SetRFRange` or `SetRFontRange` or `SetRFR`.

VTK Coding Standards

- When the name includes a natural abbreviation such as OpenGL, keep the abbreviation and capitalize the abbreviated letters.
- Keep all instance variables protected or private. The user and application developer should access instance variables through Set/Get methods. To aid in this there are a number of macros defined in `vtkSetGet.h` that can be used.

VTK Coding Standards

- Use "this->" inside of methods even though C++ doesn't require you to. This really seems to make the code more readable because it disambiguates between instance variables and local or global variables. It also disambiguates between member functions and other functions.
 - `this->Debug;` or `this->Update();`
- Common names: choose one and be consistent
 - Compute vs. Calc or Calculate

VTK Class Implementation

- Important Classes
 - vtkObjectBase – RTI (run-time interface e.g., GetClassName(), etc); reference counting and garbage collection; print interface
 - vtkObject - modification time, event processing, debug flag
 - vtkMath, vtkByteSwap, vtkMultiThreader
- Class implementation guidelines:
 - Private instance variables
 - Access through Set/Get/On/Off/Action methods
 - DebugOn / DebugOff
 - Use SetGet macros

VTK Class Implementation

- All Objects Should Implement
 - `vtkTypeRevisionMacro(class,superclass);`
 - `typedef Superclass`
 - `GetClassName()`
 - `IsTypeOf()`
 - `SafeDownCast()`
 - `NewInstance()` --- *virtual constructor*
 - `New()` – static constructor (factory method)
- Most Objects Implement
 - `PrintSelf()`
 - Constructor & Destructor

Reference Counting

- Object creation and destruction - objects shared between other objects requires careful memory management (when to delete memory?)
- In VTK: Instantiate & destroy objects with New() / Delete()
 - `vtkActor *anActor = vtkActor::New();`
 - `anActor->Delete();`
- Can not allocate on the stack due to reference counting
 - ~~`vtkActor anActor;`~~ ← *Compiler won't let you*
- If you need to hold onto an object use Register() / UnRegister() methods.
 - `anActor->Register(...);` ← *Increase reference count*
 - `anActor->UnRegister(...);` ← *Decrement reference count; same as Delete() method*

Reference Counting Example

<i>Operation</i>	<i>Result</i>	<i>this->ReferenceCount</i>
<code>*bar = vtkObject::New()</code>	<i>instantiation</i>	1
<code>bar->Register(...)</code>	<i>Increment</i>	2
<code>bar->UnRegister(...)</code>	<i>Decrement</i>	1
<code>bar->Delete()</code>	<i>Destroy</i>	0

Garbage Collector

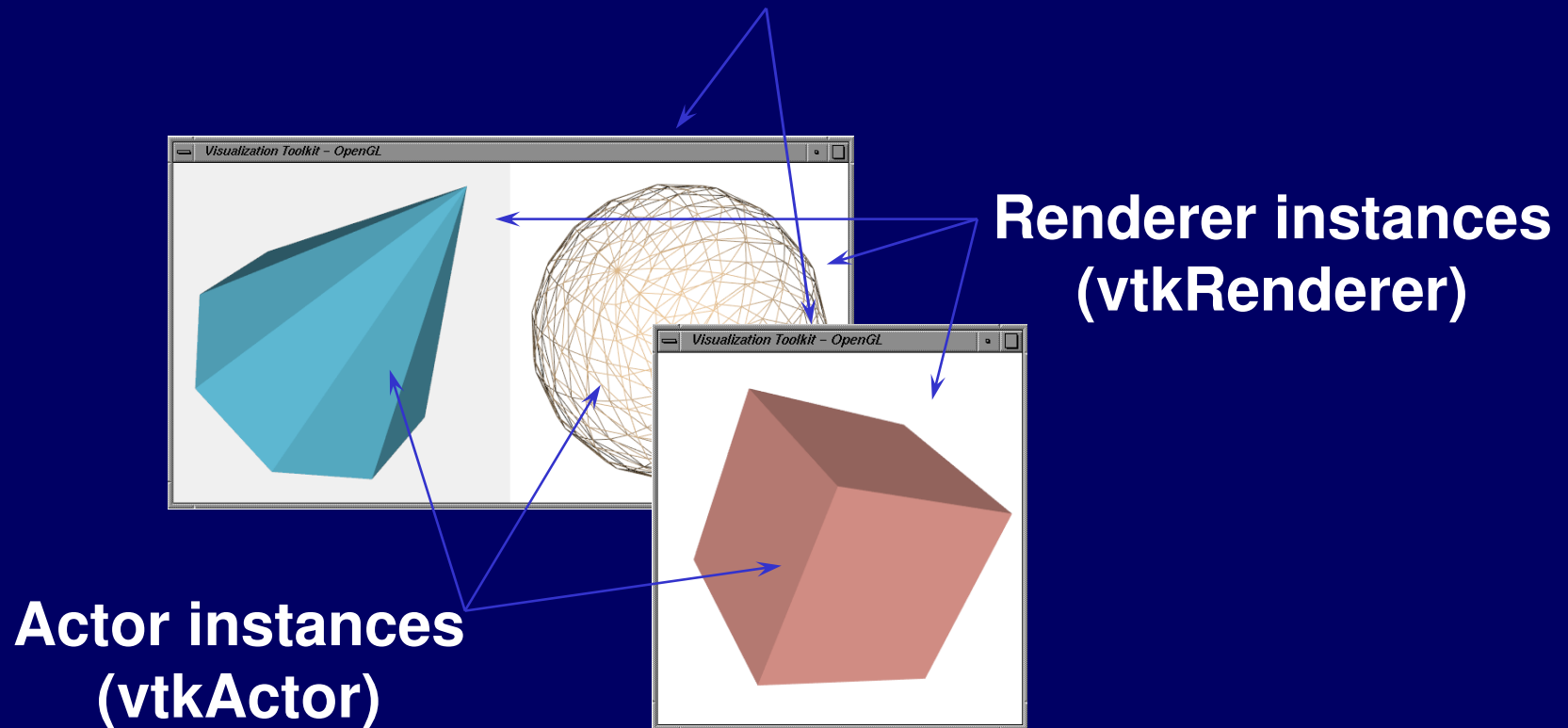
- Use to solve automatically cycling references whenever Delete() is called.
- Turned on by default. Can be turned off with:
`vtkGarbageCollector::DeferredCollectionPush()`
- Previous garbage collector status can be restored with
`vtkGarbageCollector::DeferredCollectionPop()`

Major VTK Subsystems

- Graphics
- Image processing pipeline
 - ImageData (data object)
 - Filters (process only vtkImageData)
- 3D geometry processing pipeline
 - DataSets (i.e., data objects)
 - Filters (i.e., process datasets or subclasses)
- Picking, Interaction, etc.

The VTK Graphics Subsystem

Instances of render window (`vtkRenderWindow`)



The VTK Graphics Subsystem

A VTK scene consists of:

- `vtkRenderWindow` - contains the final image
- `vtkRenderer` - draws into the render window
- `vtkActor` - combines properties / geometry
 - `vtkProp`, `vtkProp3D` are superclasses
 - `vtkProperty`
- `vtkLights` - illuminate actors
- `vtkCamera` - renders the scene
- `vtkMapper` - represents geometry
 - `vtkPolyDataMapper`, `vtkDataSetMapper` are subclasses
- `vtkTransform` - position actors

Typical Application (in C++)

```
vtkSphereSource *sphere = vtkSphereSource()::New(); // create data pipeline
vtkPolyDataMapper *sphereMapper = vtkPolyDataMapper()::New();
    sphereMapper->SetInputConnection(sphere->GetOutputPort());
vtkActor *sphereActor = vtkActor()::New();
    sphereActor->SetMapper(sphereMapper); //mapper connects actor with pipeline

vtkRenderer *renderer = vtkRenderer()::New();
vtkRenderWindow *renWin = vtkRenderWindow()::New();
    renWin->AddRenderer(renderer);
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor()::New();
    iren->SetRenderWindow(renWin);

renderer->AddViewProp(sphereActor);
renderer->SetBackground(1,1,1);
renWin->SetSize(300,300);

renWin->Render();
iren->Start(); //starts the event loop
```

Exercise 1b

Introduction

- Start Microsoft Visual C++ (or other compiler)
- Load the exercise1b workspace (or makefile)
- Compile & run the program (keypress “e” exits)
- Replace the cone with a sphere
- Set the sphere resolution

Exercise 1b (answer)

- Experiment with the resolution of the cone
 - What happens with higher values?
 - What happens when Resolution == 1?
 - What happens when Resolution == 2?
- Use a sphere rather than a cone
 - #include vtkSphereSource.
 - Instantiate a vtkSphereSource.
 - Set resolution
 - SetPhiResolution()
 - SetThetaResolution().

Graphics API Overview

- The following is a summary of instance variables & methods
- Remember there is typically a `Set__()` and `Get____()` method to set and get the instance variable values.
- Refer to Doxygen man pages, or class header files, for more information.

vtkRenderWindow

- AddRenderer() – add another renderer which draws into this vtkRenderWindow
- SetSize() – set the size of the window
- SetPosition() – set the position of the window
- SetWindowName() – set the name (in the titlebar)
- AAFrames, FDFrames, SubFrames – used for anti-aliasing and focal depth
- StereoType, StereoRenderOn/Off – control stereo
- AbortRender, AbortCheckMethod – methods to interrupt the rendering process

vtkRenderWindow (cont.)

- DesiredUpdateRate – a frame rate which is used to control LOD (level-of-detail) actors
- DoubleBuffer – turn double buffering on/off
- PixelData, RGBAPixelData, ZbufferData – set/get the color buffer and depth buffer for the window

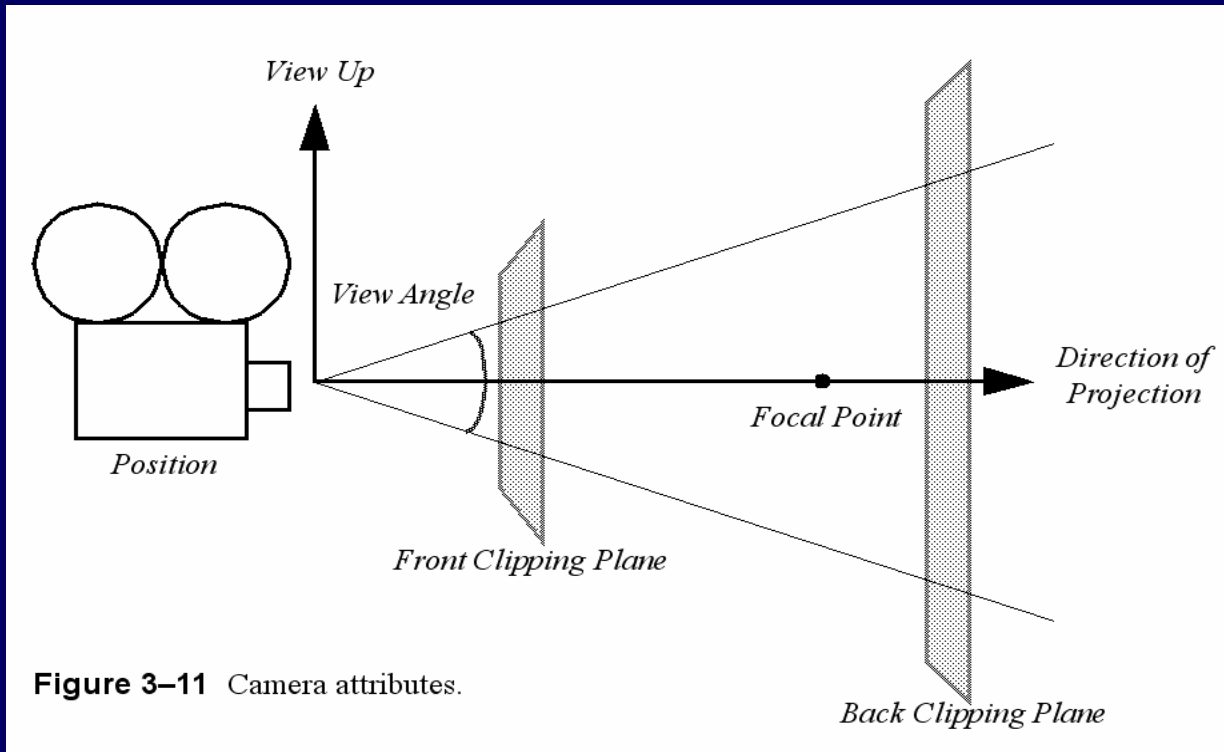
vtkRenderer

- AddViewProp (preferred), AddActor, AddVolume, AddActor2D – add objects to be rendered
- AddLight – add a light to illuminate the scene
- SetAmbient – set the intensity of the ambient lighting
- SetViewport – specify where to draw in the render window
- SetActiveCamera – specify the camera to use render the scene
- ResetCamera – reset the camera so that all actors are visible

vtkCamera

- Position – where the camera is located
- FocalPoint – where the camera is pointing
- ViewUp – which direction is “up”
- ClippingRange – data outside of this range is clipped
- ViewAngle – the camera view angle controls perspective effects
- EyeAngle – the angle between eyes (for stereo)
- ViewPlaneNormal – the normal vector to the view plane

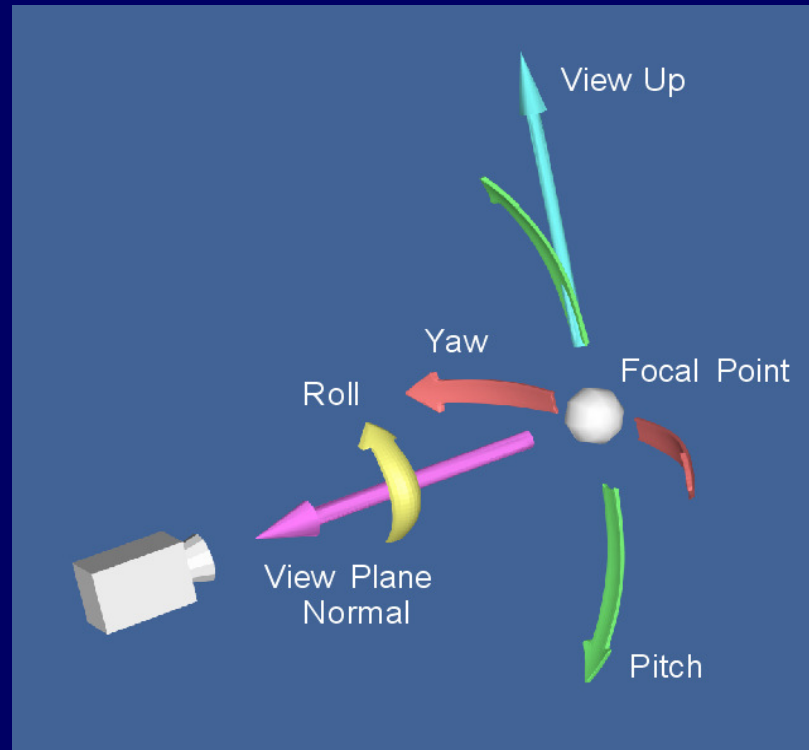
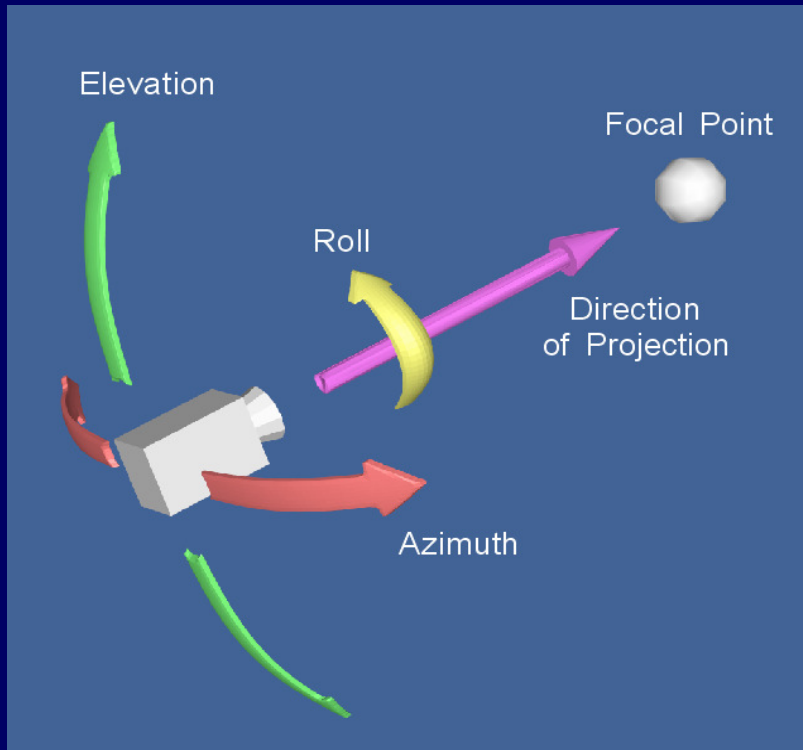
vtkCamera (cont.)



vtkCamera (cont.)

- ParallelProjection – turn parallel projection on/off (no perspective effects)
- ParallelScale – used to shrink or enlarge an image
- Roll, Pitch, Yaw, Elevation, Azimuth – move the camera in a variety of ways
- Zoom, Dolly – changes view angle (Zoom); move camera closer (Dolly)
- OrthogonalizeViewUp – make the view up vector perpendicular to the view plane normal

vtkCamera (cont.)



vtkLight

- Color – the light color
- Position – where the light is
- FocalPoint – where the light is pointing
- Intensity – the brightness of the light
- Switch – turn the light on or off
- Positional – is it an infinite or local (positional) light
- ConeAngle – the cone of rays leaving the light

vtkActor (subclass of vtkProp)

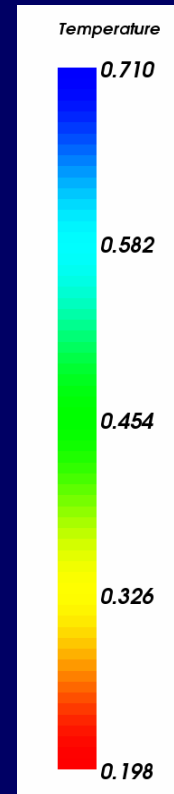
- Property – surface lighting properties
- Texture – a texture map associated with the actor
- Position – where it's located
- Origin – the origin of rotation
- Visibility – is the actor visible?
- Pickable – is the actor pickable?
- Draggable – is the actor draggable?
- RotateX, RotateY, RotateZ – rotate around different axes
- RotateWXYZ – rotate around a vector

vtkProperty

- Interpolation - shading interpolation method (*Flat, Gouraud*)
- Representation – how to represent itself (*Points, Wireframe, Surface*)
- AmbientColor, DiffuseColor, SpecularColor – a different color for ambient, diffuse, and specular lighting
- Color – sets the three colors above to the same
- Ambient, Diffuse, Specular – coefficients for ambient, diffuse, and specular lighting
- Opacity – control transparency

vtkLookupTable

- NumberOfColors – number of colors in the table
- TableRange – the min/max scalar value range to map
- If building a table from linear HSVA ramp:
 - HueRange – min/max hue range
 - SaturationRange – min/max saturation range
 - ValueRange – min/max value range
 - AlphaRange – min/max transparency range
- If manually building a table
 - Build (after setting NumberOfColors)
 - SetTableValue(idx, rgba) for each NumberOfColors entries



Example: Initial Camera View

```
vtkCamera *cam1 = vtkCamera::New();
    cam1->SetFocalPoint( 0, 0, 0 );
    cam1->SetPosition( 1, 1, 1 );
    cam1->SetViewUp( 1, 0, 0 );
    cam1->OrthogonalizeViewUp();

ren1->SetActiveCamera( cam1 );
ren1->ResetCamera();
```

Exercise 1c

- Modify `exercise1c.cxx` to:
 - Create initial view down the $(1,1,1)$ vector
 - Set ambient, diffuse, specular actor color
 - Make the actor have a wireframe representation
 - Change the background color of the Renderer

Exercise 1c (answer)

```
// work the the actor's property.
// one is created by default if a property has not been specified
vtkProperty *prop = actor1->GetProperty();
prop->SetDiffuseColor(0,0,1.0);
prop->SetSpecularColor(0.0,1.0,0.0);
    prop->SetSpecular(1);
    prop->SetSpecularPower(10);
    prop->SetAmbientColor(1,0,0);
    prop->SetAmbient(0.3);
//prop->SetRepresentationToWireframe();

// add the actor to the renderer and start handling events
aren->AddViewProp(actor1);
aren->SetBackground(0.1,0.2,0.4);

// setup a default camera view
aren->GetActiveCamera()->SetFocalPoint(0,0,0);
aren->GetActiveCamera()->SetPosition(1,1,1);
aren->GetActiveCamera()->SetViewUp(0,1,0);
aren->GetActiveCamera()->OrthogonalizeViewUp();
aren->ResetCamera();

renWin->Render();
iren->Start();
```

Important vtkProp Subclasses

- vtkLODActor - automated LOD creation
- vtkLODProp3D - manual control of LOD's including mixed volumes/surfaces
- vtkFollower - always face a camera
- vtkAssembly - groups of vtkProp3D's, transformed together.

vtkLODActor

- Changes resolution based on desired response
- ```
vtkLODActor *actor = vtkLODActor::New();
actor->SetMapper(mapper);
actor->SetNumberOfCloudPoints(1000);
```
- ```
vtkRenderWindow *renWin =  
    vtkRenderWindow::New();  
renWin->SetDesiredUpdateRate( 5.0 );
```

vtkLODProp3D

- `vtkLODProp3D lod`

```
lod AddLOD volumeMapper volumeProperty2 0.0
lod AddLOD volumeMapper volumeProperty 0.0
lod AddLOD probeMapper_hres probeProperty 0.0
lod AddLOD probeMapper_lres probeProperty 0.0
lod AddLOD outlineMapper outlineProperty 0.0
```
- *From `Examples/VolumeRendering/Tcl/volSimpleLOD.tcl`*

vtkFollower

Actor always faces a specified camera

```
vtkFollower *textActor = vtkFollower::New();  
textActor->SetMapper( textMapper );  
textActor->SetScale( 0.2, 0.2, 0.2 );  
textActor->AddPosition( 0, -0.1, 0 );  
textActor->SetCamera( aCamera );
```

vtkAssembly

Create hierarchies of vtkProp3D's:

```
vtkAssembly *cylinderActor = vtkAssembly::New();
    cylinderActor->AddPart( sphereActor );
    cylinderActor->AddPart( cubeActor );
    cylinderActor->AddPart( coneActor );
    cylinderActor->SetOrigin( 5, 10, 15 );
    cylinderActor->AddPosition( 5, 0, 0 );
    cylinderActor->RotateX( 15 );
```

vtkRenderWindowInteractor

Key features:

- SetRenderWindow – the single render window to interact with
- Key and mouse bindings (Interactor Style)
- Light Follow Camera (a headlight)
- Picking interaction

Interactor Style(s)

- Button 1 – rotate
- Button 2 – translate (<Shift> Button 1 for 2-button mouse)
- Button 3 – zoom
- Keypress e or q – exit
- Keypress f – “fly-to” point under mouse
- Keypress s/w – surface/wireframe
- Keypress p – pick
- Keypress r – reset camera
- Keypress 3 – toggle stereo

Switch styles: Keypress j – joystick; t - trackball style

Picking

- `vtkPropPicker` - *hardware-assisted* picking of `vtkProps` (returns `vtkProp` picked and x,y,z coordinate)
- `vtkWorldPointPicker` - get x-y-z coordinate; does not pick prop (*hardware assisted*, returns x,y,z coordinate)
- `vtkPicker` - pick based on prop3D's bounding box (*software* ray cast – returns `vtkProp`)
- `vtkPointPicker` - pick points (closest point to camera within tolerance - *software* ray cast – returns point id & x,y,z coordinate)
- `vtkCellPicker` - pick cells (*software* ray cast – returns cell id & x,y,z)

Example: Picking and Style

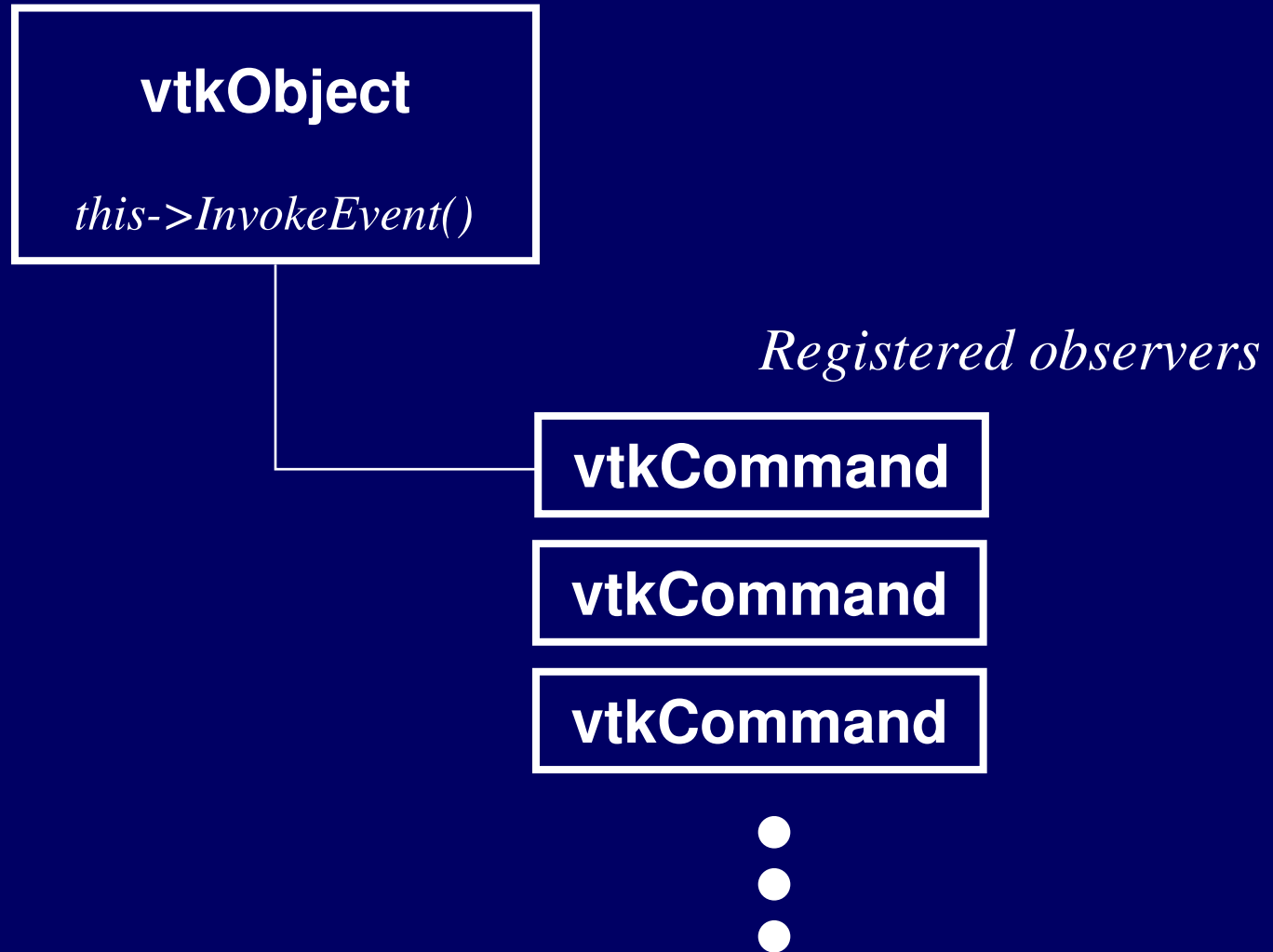
```
vtkRenderWindowInteractor *iren =  
    vtkRenderWindowInteractor::New();  
vtkInteractorStyleFlight *style =  
    vtkInteractorStyleFlight::New();  
vtkCellPicker *picker = vtkCellPicker::New();  
  
iren->SetInteractorStyle(style);  
iren->SetPicker(picker);
```

(Note: defaults are automatically created, you rarely ever need to do this)

Command / Observer Callbacks

- Observers observe events on a particular VTK instance
 - Use `AddObserver()` to watch for events
 - The observer is implicit; you actually add the command; e.g. `vtkObject::AddObserver(unsigned long event, vtkCommand*)`
- When an observer sees the event it is interested in, it invokes a command via the command's `Execute()` method
- The events originate from instances that invoke events on themselves (and may pass call data)
 - `this->InvokeEvent(vtkCommand::ProgressEvent, NULL);`

Command / Observer



VTK Events (Command & Observer Design Pattern)

- `vtkObject` supports invoking events listed in `vtkCommand::EventIds`:
 - `void vtkObject::InvokeEvent(unsigned long event, void *callData)`
 - `void vtkObject::InvokeEvent(const char *event, void *callData)`
- Observers can be attached to any VTK object for any listed event (any subclass of `vtkObject`):
 - `unsigned long vtkObject::AddObserver(unsigned long event, vtkCommand*)`
 - `unsigned long vtkObject::AddObserver(const char *event, vtkCommand*)`
- When receiving an event, observers call the virtual method `Execute()` on their commands:
 - `virtual void vtkCommand::Execute(vtkObject *caller, unsigned long eid, void *callData) = 0`

VTK Events (2)

- An observer takes action via a subclass of `vtkCommand` (e.g. `vtkCallbackCommand` or a class that you derive-*preferred method*)
- Subclass `vtkCommand` and implement an `Execute()` method
- There are already some general purpose observers
 - `vtkCallbackCommand`
 - `vtkOldStyleCallbackCommand` - *legacy*
 - See `VTK/Common/vtkCommand.h`

Creating Callbacks – Two Approaches

- Deriving your own class from `vtkCommand`
 - Requires implementing `Execute()` method
 - *Preferred method*
- Use `vtkCallbackCommand`
 - Requires creating a function of the form:

```
void func(vtkObject *caller, unsigned long  
eid, void* clientdata, void *calldata)
```
 - Set the callback with `SetCallback()`
 - Set any client data (associated with callback)

Approach #1 - Derive A Command

```
class myCallback : public vtkCommand
{
public:
    static myCallback *New() { return new myCallback; }
    virtual void Execute(vtkObject *caller, unsigned long eid,
        void *callData)
    {
        vtkRenderer *r = (vtkRenderer *)caller;
        cerr << "Starting to Render with renderer" << *r << "\n";
    }
};

myCallback *c = myCallback::New();
ren1->AddObserver(vtkCommand::StartEvent, c);
c->Delete(); //okay, reference counting
```

Approach #2 - vtkCallbackCommand

- Define function to invoke

```
void Picked(vtkObject*, unsigned long, void *clientdata, void
*calldata)
{
    vtkCellPicker *picker = (vtkCellPicker *)clientdata;
    cerr << "Picked cell id " << picker->GetCellId() << endl;
}
```

- Instantiate vtkCallbackCommand

```
vtkCallbackCommand *cmd = vtkCallbackCommand::New();
cmd->SetCallback(Picked);
cmd->SetClientData(picker);
```

- AddObserver()

```
iren->AddObserver(vtkCommand::EndPickEvent, cmd);
```

Determine Which Events Are Invoked by Which Classes

- Doxygen Documentation
 - Description section, look for “Events”
 - Related Pages (Class to Events, Event to Classes)
- Source code “grep” or “find”
- Don't forget superclass

Exercise 1d

Modify `exercised.cxx` to:

- Create a `vtkCellPicker`
- Assign it to `vtkRenderWindowInteractor`
- After picking, print out `cellId`
- (*Hint: see `VTK/Examples/Tutorial/Step2/Cxx/Cone2.cxx`*)

Exercise 1d (solution)

```
class vtkMyCommand : public vtkCommand
{
public:
    static vtkMyCommand* New() { return new vtkMyCommand; }
    virtual void Execute(vtkObject *caller, unsigned long eventId,
                        void *callData)
        {vtkCellPicker *picker = vtkCellPicker::SafeDownCast(caller);
         cerr << "Picked cell id " << picker->GetCellId() << endl; }
};

main ()
{
    vtkMyCommand *cmd = vtkMyCommand::New();
    vtkCellPicker *picker = vtkCellPicker::New();
    picker->AddObserver(vtkCommand::EndPickEvent, cmd);

    vtkRenderer *aren = vtkRenderer::New();
    vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(aren);
    iren->SetPicker(picker);
}
```

Exercise 1d (alternate answer)

```
void Picked(vtkObject*, unsigned long, void *arg, void*)
{
    vtkCellPicker *picker = (vtkCellPicker *)arg;
    cerr << "Picked cell id " << picker->GetCellId() << endl;
}

main ()
{
    vtkCellPicker *picker = vtkCellPicker::New();

    vtkCallbackCommand *cmd = vtkCallbackCommand::New();
    cmd->SetCallback(Picked);
    cmd->SetClientData(picker);

    vtkRenderer *aren = vtkRenderer::New();
    vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(aren);

    vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWin);
    iren->SetPicker(picker);
    iren->AddObserver(vtkCommand::EndPickEvent, cmd);
    ...
}
```

3D Widgets

- Added since VTK 4.0 release
 - Requires version 4.2 or later
- Subclass of `vtkInteractorObserver`
 - Interactor observers watch events invoked on `vtkRenderWindowInteractor`
 - Events are caught and acted on
 - Events can be prioritized and ordered
 - The handling of a particular event can be aborted

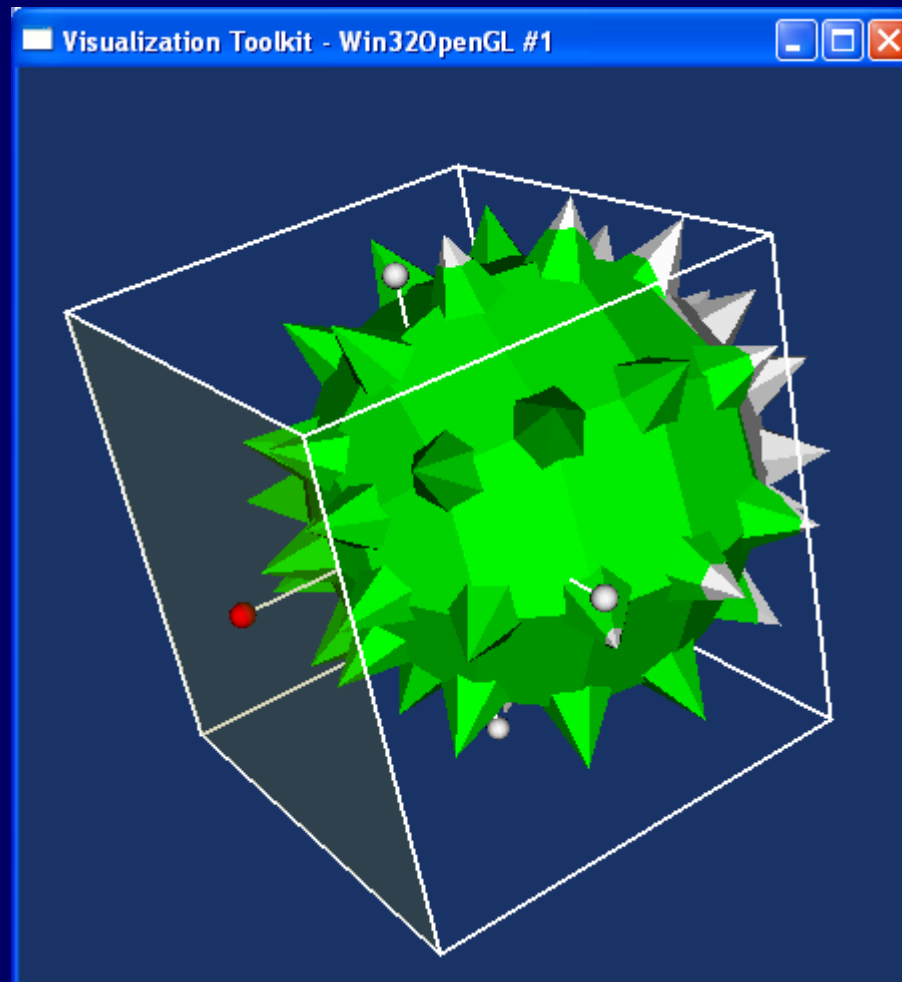
Some 3D Widgets

- `vtkPointWidget`
- `vtkLineWidget`
- `vtkPlaneWidget`
- `vtkImplicitPlaneWidget`
- `vtkImagePlaneWidget`
- `vtkBoxWidget`
- `vtkSphereWidget`
-

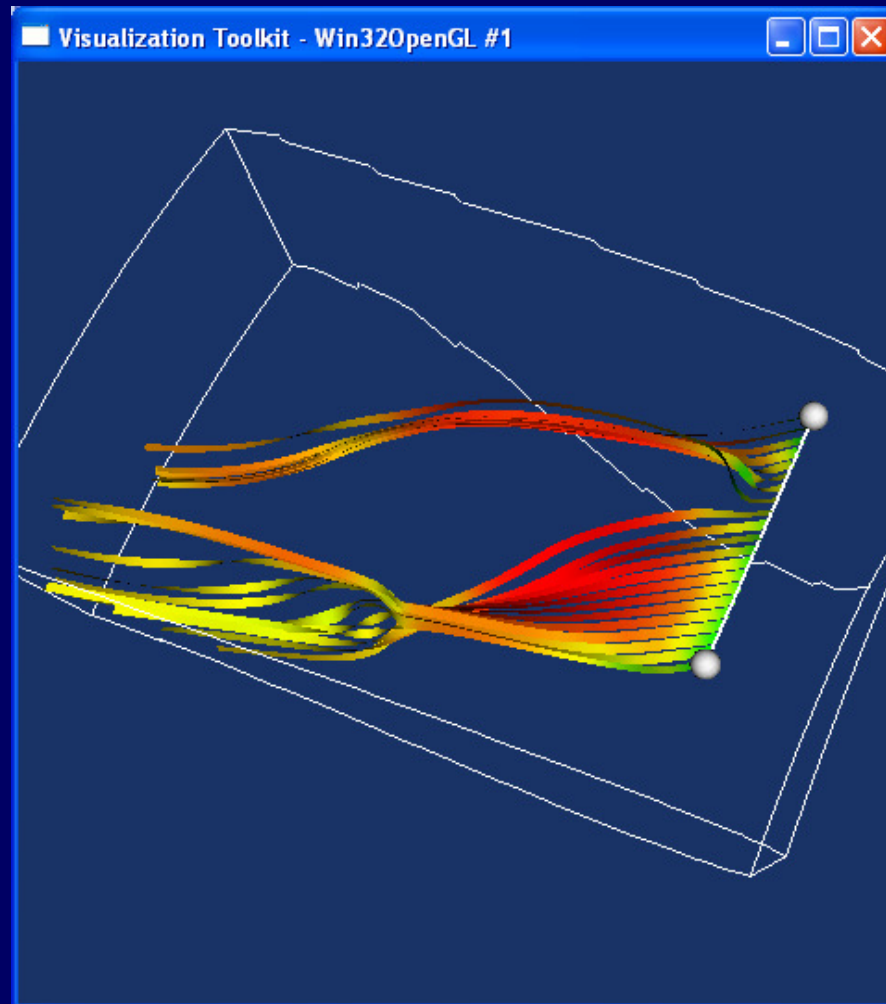
- Widgets often provide auxiliary functionality (e.g., obtaining transforms, polydata, implicit functions, etc.)

- More than one widget at a time can be used

vtkBoxWidget



vtkLineWidget



Example Usage

```
vtkLineWidget lineWidget
vtkPolyData seeds
lineWidget SetInput [p13d GetOutput]
lineWidget SetAlignToXAxis
lineWidget PlaceWidget
lineWidget GetPolyData seeds

.....
.....

lineWidget SetInteractor iren
lineWidget AddObserver StartInteractionEvent
BeginInteraction
lineWidget AddObserver InteractionEvent
GenerateStreamlines
```

Exercise 1e (Optional)

- Compile and run
 - `BoxWidget.cxx`
 - `ImagePlaneWidget.cxx`
 - *Hint: remember to type “i” to realize widget*
 - *Note: Tcl variants of the examples are also available*
- Study and understand the code