Today’s Class

- Highlights from HW #2
- This Week’s Readings
- Next Week’s Readings
- Immersive Interactive Environments in EMPAC
- VTK Interaction
  - Callbacks
  - Subclassing Interactors
  - Mouse & Key Events
  - Picking

Interaction & Picking

Real 3D Coordinates

Simplicity

Density of Information? Importance of Labels?

Tricks to Visualizing Multiple Edges?
Today’s Class

- Highlights from HW #2
- This Week’s Readings
- Next Week’s Readings
- Immersive Interactive Environments in EMPAC
- VTK Interaction
  - Callbacks
  - Subclassing Interactors
  - Mouse & Key Events
  - Picking

Readings for This Week:

- “Graph drawing by force-directed placement”, Fruchterman & Reingold, 1991

Readings for This Week:

- “Heapviz: Interactive Heap Visualization for Program Understanding and Debugging” Aftandilian, Kelley, Gramazio, Ricci, Su, & Guyer, 2010
Today’s Class

• Highlights from HW #2
• This Week’s Readings
• Next Week’s Readings
• Immersive Interactive Environments in EMPAC
• VTK Interaction
  – Callbacks
  – Subclassing Interactors
  – Mouse & Key Events
  – Picking

Readings for Next Week:

• “Focus Plus Context Screens: Combining Display Technology with Visualization Techniques”, Baudisch, Good, & Stewart, UIST 2001

Readings for Next Week:

• “Visualization of Exception Handling Constructs to Support Program Understanding”, Shah, Gorg, & Harrod, Software Visualization 2008

Today’s Class

• Highlights from HW #2
• This Week’s Readings
• Next Week’s Readings
• Immersive Interactive Environments in EMPAC
• VTK Interaction
  – Callbacks
  – Subclassing Interactors
  – Mouse & Key Events
  – Picking

Our System Goals/Requirements

• Large, human-scale projection environment
• People move freely within the space
• Projection surfaces can be moved interactively
• Varying illumination conditions
• Robust & real-time tracking and display

Dynamic Projection Surfaces for Immersive Visualization

Theodore C. Yapo, Yu Sheng, Joshua Nasman, Andrew Dolce, Eric Li, and Barbara Cutler
PROCAMS 2010 IEEE International Workshop on Projector-Camera Systems, June 2010
Architectural Daylighting Design
- Windows, wall colors, & time of day
  controlled through iTouch interface

Volumetric Visualization
- Cross sections of a 3D medical dataset
  virtually placed within the projection volume

General User Interface Elements
- Projection surfaces as input devices
- No instruction necessary to play the game!

Today’s Class
- Highlights from HW #2
- This Week’s Readings
- Next Week’s Readings
- Immersive Interactive Environments in EMPAC
- VTK Interaction
  – Callbacks
  – Subclassing Interactors
  – Mouse & Key Events
  – Picking

Introduction to VTK: Interaction
What is Interaction?

• Manipulating the objects in a scene
  o Moving
  o Rotating
  o Selecting
  o Deleting

• Manipulating your view of the scene (manipulating the camera)
  o Pan
  o Tilt
  o Zoom

Interaction Devices

• Keyboard
  o Press a key
• Joystick
• Mouse
  o Left button
  o Middle button
  o Right button
  o Single click
  o Double click
• 3D mouse (3D Connexions Space Navigator)

Manipulating the Camera

Interaction in VTK

• Callback functions
  o Less overhead
  o Less powerful
• Subclass an existing vtkInteractorStyle
  o Preferred method

Callback Functions

```c
void ClickCallbackFunction ( vtkObject* caller, long unsigned int eventId, void* clientData, void* callData )
{
  caller – The object that called the callback.
  eventId – What happened to trigger this function?
  clientData – Give the callback access to an object.
  callData – Additional information specific to this particular call.
}
```

Callback Functions (cont.)

```c
void RefreshCallback ( vtkObject* caller, long unsigned int eventId, void* clientData, void* callData )
{
  // Get the object that called the callback
  vtkRenderWindow* ren = static_cast<vtkRenderWindow*>(caller);
  vtkActor* actor = static_cast<vtkActor*>(clientData);
  // Access anything you put in clientData
  vtkActor* actor = static_cast<vtkActor*>(clientData);
}
```
Connecting the Callback

```plaintext
vtkCallbackCommand* refreshCallback =
vtkCallbackCommand::New();
refreshCallback->SetCallback(RefreshCallback); // The name of the function you have defined above
refreshCallback->SetClientData(actor);
renderWindow->AddObserver(vtkCommand::ModifiedEvent,refreshCallback);
```

Observable Events in VTK

- LeftButtonPressEvent
- LeftButtonReleaseEvent
- MiddleButtonPressEvent
- MiddleButtonReleaseEvent
- RightButtonPressEvent
- RightButtonReleaseEvent
- KeyPressEvent
- KeyReleaseEvent
- MousePressEvent
- Many more (see EventIds enum here

```
• http://www.vtk.org/Wiki/VTK/Examples/Cxx/Interaction/KeypressObserver
• http://www.vtk.org/Wiki/VTK/Examples/Cxx/Interaction/MouseEventsObserver
```

Callback Functions Examples

```
• http://www.vtk.org/Wiki/VTK/Examples/Cxx/Interaction/KeypressObserver
• http://www.vtk.org/Wiki/VTK/Examples/Cxx/Interaction/MouseEventsObserver
```

vtkInteractorStyle

- Abstract class to catch user actions
- Subclasses implement particular functionalities
- Subclasses included in VTK:
  - vtkInteractorStyleTrackballActor
  - vtkInteractorStyleTrackballCamera
  - vtkInteractorStyleRubberBand2D
  - Many more (see http://www.vtk.org/doc/nightly/html/classes.html)

```
• http://www.vtk.org/Wiki/VTK/Examples/Cxx/Visualization/MoveActor
• http://www.vtk.org/Wiki/VTK/Examples/Cxx/Visualization/MoveCamera
```

Existing InteractorStyle Examples

```
• vtkInteractorStyleTrackballActor
  http://www.vtk.org/Wiki/VTK/Examples/Cxx/Visualization/MoveActor
• vtkInteractorStyleTrackballCamera
  http://www.vtk.org/Wiki/VTK/Examples/Cxx/Visualization/MoveCamera
```

Custom Interaction - Subclassing InteractorStyles

- The idea is to simply re-implement functions like OnKeyPress, OnLeftButtonDown, etc with your own functionality
- There is some overhead because of VTK's structure
class CustomStyle : public vtkInteractorStyleTrackballCamera
{
public:
    static CustomStyle* New();
    vtkTypeMacro(CustomStyle, vtkInteractorStyleTrackballCamera);
    virtual void OnLeftButtonDown()
    {
        std::cout << "Pressed left mouse button." << std::endl;
        // Forward events
        vtkInteractorStyleTrackballCamera::OnLeftButtonDown();
    }
    vtkStandardNewMacro(CustomStyle);
};

vtkRenderWindowInteractor* renderWindowInteractor = vtkRenderWindowInteractor::New();
renderWindowInteractor->SetRenderWindow(renderWindow);
CustomStyle* style = CustomStyle::New();
renderWindowInteractor->SetInteractorStyle(style);

Custom Interaction - Subclassing InteractorStyles (cont.)

Using Your New Interactor

• Just like before!

Picking

• Get the world coordinates of a mouse click
  • vtkPropPicker

Picking (cont.)

virtual void OnLeftButtonDown()
{
    // Get the screen/window/pixel coordinates
    int* clickPos = this->GetEventPosition();
    // Pick from this location
    vtkSmartPointer<vtkPropPicker> picker = vtkSmartPointer<vtkPropPicker>::New();
    picker->Pick(clickPos[0], clickPos[1], 0, this->GetDefaultRenderer());
    // Get the world coordinates
    double* pos = picker->GetPickPosition();
    std::cout << "Pick position (world coordinates) is: " << *pos[0] << *pos[1] << *pos[2] << std::endl;
}
Object Intersection

- VTK does not have much to offer
- There is a nice "add on" ([http://www.midasjournal.org/browse/publication/726](http://www.midasjournal.org/browse/publication/726)) that wraps GTS (GNU Triangulated Surface Library)
- What you can do is "Point inside object" testing

Point Inside Object

```cpp
tkSelectEnclosedPoints* selectEnclosedPoints =
tkSelectEnclosedPoints*: New();
selectEnclosedPoints->SetInput(transformPolyData->GetOutput());
selectEnclosedPoints->SetSurface(this->Sphere);
selectEnclosedPoints->Update();
```

```cpp
vtkDataArray* insideArray =
vtkDataArray::SafeDownCast(
    selectEnclosedPoints->GetOutput()-> GetPointData()-> GetArray("SelectedPoints"));
```

(Point Inside Object (cont.))

```cpp
bool inside = false;
for(vtkIdType i = 0; i < insideArray->GetNumberOfTuples(); i++)
{
    if(insideArray->GetComponent(i,0) == 1)
    {
        inside = true; break;
    }
}
```

Positioning an Actor (round 2)

- In a previous class we learned about `vtkActor::SetUserTransform`
- However, when you interact with an actor, other internal stat variables (Position and Orientation) are modified.
- The UserTransform is generated from the Position and Orientation, so if you REALLY want to set the position/orientation, use:

```cpp
this->CubeActor->SetPosition(0,0,0);
this->CubeActor->SetOrientation(0,0,0);
```

Intersection Example