Visualization History

- Visualization libraries
  - Embed into simulation code
  - Outputs images (plots) during run-time.
- Post-process of restart dumps
  - Custom tools for custom data formats
  - Custom derived data, data mappers

Visual Programming

- Haberli’s landmark ConMan paper -1985?
- Extension of Unix pipes.
- Take a small specialized program
  - Provide wrapper for remote procedure and shared memory
  - Provide GUI for parameters (args) of the program.

Early Visual Prog. Env.

- Data Push models
  - aPe - Ohio State / OSC
  - AVS - Advanced Visualization System
    - Stellar Computer
  - Khoros - Univ. of New Mexico?
  - Iris Explorer - SGI / NAG

- Data Flow model

Great concepts

- Very similar to Corba support
  - provides the shared memory communication
  - each “module” is its own process
    - Can reside on its own CPU
    - Can reside across the network
- Tools to automatically convert a C, C++ or Fortran program to a module.
Problems

- Data pushed to module A
- Data modified by A (a “mapper”) and pushed to both modules B and C.
- Four copies of the data now exist

Execution Model

- Each module has a set of parameters or state that controls its output.
- A change to Module B’s parameters only requires re-executing B.
- A change to Module A’s parameters, will re-execute A and send new data to both B and C.

Data Push vs Data Pull

- Data Push
  - Push data when changed
- Data Pull
  - Only request data when needed
  - Break the data into chunks, and only request small chunks at a time.
- Multiple windows (paths)
- Memory requirements reduced

Early Visual Prog. Env.

- Data Pull Models
  - Data Explorer (Dx) - IBM
    - Data centric
  - AVS/Express
  - Image Vision - SGI
  - Java Advanced Imaging
  - Vtk
  - Much more complicate, Much more flexible
VTK
The Visualization Toolkit

William J. Schroeder
President, Kitware, Inc.
Rensselaer SCOREC Research Associate

www.kitware.com
www.vtk.org

will.schroeder@kitware.com

Themes

• Overview
• Architecture
• Graphic Subsystem
• Visualization Pipeline
• Interaction
• Software Process
• Conclusion

Themes

Taxonomy of Visualization Systems

• Toolkits / Libraries
  - Components
  - Little or no GUI
  - Meant to be integrated with other systems/toolkits
  - Programming required

• Development Environments
  - Components
  - GUI-assisted (visual) programming

• Applications
  - Turn-key
  - Minimal programming

What Is VTK?

A visualization toolkit

- Designed and implemented using object-oriented principles
- C++ class library (~700+ classes; ~425,000 LOC; <225,000 executable lines)
- Automated Java, TCL, Python bindings
- Portable across Unix, Windows, Mac OSX
- Supports 3D/2D graphics, visualization, image processing, volume rendering
- Focused on scientific visualization, some support for higher-dimensional visualization
- Open-source (BSD-style, not GPL)
- www.vtk.org
VTK Is A Toolkit, Not a System

- Embeddable
  - Plays with other software
- Separable
  - Can pull out "pieces"
- Adaptable
  - Not dependent on GUI
  - Not dependent on rendering library
  - OpenGL, others

History

- Precursor: GE Legacy System LYMB (circa 1983)
- Proprietary software issues led to open-source VTK project to create:
  - Textbook
  - Sample code
- December '93 work begins on VTK
- June 1998 Kitware forms:
  - VTK Commercial Support

More Things: Image-Processing

Things It Does

- Image-Processing
Volume Rendering

Functionality

- **Graphics / Modeling**
  - Smoothing, decimation, extrusion, Delaunay triangulation (2D/3D), etc.
- **Image Processing**
  - 100's of filters
  - Integrated 2D / 3D
- **Visualization**
  - Scalar, vector, tensor fields (100's of filters)
  - Probing, cutting, clipping, extraction
  - Volume rendering
- **Multi-dimensional**
  - Projection, resampling

Combined Volume/Surface Rendering

How To Use It

- **Choose programming language**
  - C++
  - Tcl
  - Python
  - Java
- **Assemble components to build applications**
Typical Application (in C++)

```cpp
vtkSphereSource *sphere = vtkSphereSource::New();

vtkPolyDataMapper *sphereMapper = vtkPolyDataMapper::New();
sphereMapper->SetInput(sphere); // create data pipeline

vtkActor *sphereActor = vtkActor::New();
sphereActor->SetMapper(sphereMapper); // mapper connects the actor with pipeline

vtkRenderer *renderer = vtkRenderer::New();
renderer->AddActor(sphereActor);
renderer->SetBackground(1,1,1);
renderer->AddActor(renWin); // graphics library independent

tkWin = vtkRenderWindow::New();
tkWin->AddRenderer(renderer); // create graphics window

renWin->Start(); // being event loop
```

VTK Architecture

- Compiled C++ core
- Performance
- Code correctness
- Wrapper layer
- Often interpreted (e.g. Tcl)
- Can be compiled layer
- Microsoft COM

Architecture

- Compiled / Interpreted
  - C++ core
  - C++, Python, Java wrappers
- Subsystems
  - Graphics
  - Visualization pipeline
  - Image processing
  - 3D data processing
  - Interaction

Tcl Interpreter

- source vtkInt.tcl (define interpreter GUI)

Wrappers

- C++ core
- Wrapper layer
- Wrapper layer generated automatically during build process
- Often interpreted (e.g. Tcl)
- Can be compiled layer
- Microsoft COM
Code Comparison

- C++
  ```cpp
tkShrinkFilter *sf;
  sf = vtkShrinkFilter::New();
  sf->SetShrinkFactor(0.8);
  vtkPolyDataMapper *m;
  m = vtkPolyDataMapper::New();
  m->SetInput [ sf->GetOutput ]
  ```

- Tcl
  ```tcl
tkShrinkFilter sf
  sf SetShrinkFactor 0.8
  vtkPolyDataMapper m
  m SetInput [ sf GetOutput ]
  ```

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
  - `vtkPolyDataGeometry`, `vtkDataSetMapper` are subclasses
- `vtkTransform` - position actors

Graphics Model

Instances of render window (`vtkRenderWindow`)

Renderer instances (`vtkRenderer`)

Actor instances (`vtkActor`)
Visualization Pipeline

- Data objects
- Process objects
- Pipelines
- Managing execution

The Visualization Pipeline

A sequence of *process objects* that operate on *data objects* to generate geometry that can be rendered by the graphics engine or written to a file.

Visualization Model

- Data Objects
  - represent data
  - provide access to data
  - compute information particular to data (e.g., bounding box, derivatives)
- Process Objects
  - Ingest, transform, and output data objects
  - represent visualization algorithms

vtkDataObject / vtkDataSet

- *vtkDataObject* represents a “blob” of data
  - contain instance of *vtkFieldData*
  - an array of arrays
  - no geometric/topological structure
  - Superclass of all VTK data objects
- *vtkDataSet* has geometric/topological structure
  - Consists of geometry (points) and topology (cells)
  - Has associated “attribute data” (e.g., scalars, vectors) as well as field data
  - Convert data object to data set with *vtkDataObjectToDataSetFilter*
A dataset is a data object with structure.

- Structure consists of:
  - Points (x-y-z coordinates)
  - Cells (e.g., polygons, lines, voxels) that are defined by connectivity list referring to points ids
  - Access is via integer ID
  - Implicit representations
  - Explicit representations

vtkDataSet also has point and cell attribute data:
- Scalars - (multi-component)
- Vectors - 3-vector
- Tensors - 3x3 symmetric matrix
- Normals - unit vector
- Texture Coordinates 1-3D
- Array of arrays (i.e. FieldData)
** Scalars (An Aside) 
- Scalars are represented by a `vtkDataArray`
- Scalars are typically single valued
- Scalars can also represent color
  - \( I \) (intensity)
  - \( IA \) (intensity-alpha: \( alpha \) is opacity)
  - RGB (red-green-blue)
  - RGBA (RGB + alpha)
- Scalars can be used to generate colors
  - mapped through lookup table
  - if unsigned char \( \rightarrow \) direct color specification

** Process Objects 
- Process objects operate on data objects

```
Source           \( \rightarrow \) 1 or more inputs
\( \rightarrow \) 1 or more outputs
\( \downarrow \) Filter
\( \rightarrow \) 1 or more outputs
\( \downarrow \) Mapper
```

** Pipeline Execution Model 
- `direction of data flow (via Execute())`
- `direction of update (via Update())`

** Creating Pipeline Topology 
- \( aFilter->SetInput(bFilter->GetOutput()) \);
- The Role of Type-Checking
  - `SetInput()` accepts dataset type or subclass
  - C++ compile-time checking
  - Interpreter run-time checking
Example Pipeline

- Decimation, smoothing, normals
- Implemented in C++

Note: data objects are not shown → they are implied from the output type of the filter

Create Reader & Decimator

```cpp
vtkBYUReader *byu = vtkBYUReader::New();
byu->SetGeometryFileName("/vtkdata/fran_cut.g");

vtkDecimatePro *deci = vtkDecimatePro::New();
deci->SetInput(byu->GetOutput());
deci->SetTargetReduction(0.9);
deci->PreserveTopologyOn();
deci->SetMaximumError(0.0002);
```

Smother & Graphics Objects

```cpp
vtkSmoothPolyDataFilter *smooth = vtkSmoothPolyDataFilter::New();
smooth->SetInput(deci->GetOutput());
smooth->SetNumberOfIterations(20);
smooth->SetRelaxationFactor(0.05);

vtkPolyDataNormals *normals = vtkPolyDataNormals::New();
normals->SetInput(smooth->GetOutput());

vtkPolyDataMapper *cyberMapper = vtkPolyDataMapper::New();
cyberMapper->SetInput(normals->GetOutput());

vtkActor *cyberActor = vtkActor::New();
cyberActor->SetMapper(cyberMapper);
cyberActor->GetProperty()->SetColor(1.0, 0.49, 0.25);
cyberActor->GetProperty()->SetRepresentationToWireframe();
```

More Graphics Objects

```cpp
vtkRenderer *ren1 = vtkRenderer::New();

vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->AddRenderer(ren1);

vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);
ren1->AddActor(cyberActor);
ren1->SetBackground(1, 1, 1);
renWin->SetSize(500, 500);
iren->Start();
```
Results

Before (52,260 triangles)

After Decimation and Smoothing (7,477 triangles)

Parallel Processing

- Why? - we have to
- Shared memory
- Distributed execution
  - MPI
  - Abstracted to support other communication
- Parallel Mechanisms
  - Task
  - Data
  - Pipeline
- Scalable
  - Petabyte example on 2048 LANL machine (Martin et al. IEEE CG&A July 2001)
    (A petabyte is 2 to the 50th power (1,125,899,906,842,624) bytes. A petabyte is equal to 1,024 terabytes.)

Examples

- Modeling turbulence (Ken Jansen Rensselaer)
  - 8.5 million tetrahedra (per time step)
  - 150 million tetrahedra (soon)

Large Data Management

- Visible Woman CT Data
  870 MBytes 1734 Slices at 512x512x2

- Bell-Boeing V-2 2 tiltrotor
  140 Gbytes
Streaming Pipeline

Basic idea: handle any size data on any size computer
- Data is broken into pieces, and pieces processed one at a time through pipeline
- Piece size based on memory limits
- Can avoid system swap
- Supports parallel processing
- Issues
  - How to create pieces
  - Mapping output from input
  - Results invariance

Interaction

Command / Observer callback mechanism
- Interactor Styles
  - Key features:
    - SetRenderWindow - the single render window to interact with
    - Key and mouse bindings (Interactor Style)
    - Light Follow Camera (a headlight)
    - Picking interaction
- 3D Widgets
- Filter

Pieces

Imaging: i,j,k extents.
Graphics

xMin xMax
yMin
zMin
zMax
image->SetUpdateExtent(0, 128, 0, 128, 0, 90);
image->Update();
data->SetUpdateExtent(0, 2);
data->Update();
Piece = 0
NumberOfPieces = 2

vtkRenderWindowInteractor
Key features:
- 3D Widgets
**Interactor Style(s)**

- Button 1 - rotate
- Button 2 - translate (<Shift> Button 1 on PC)
- 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
- vtkPicker - pick based on prop3D’s bounding box (software ray cast)
- vtkPointPicker - pick points (closest point to camera within tolerance - software ray cast)
- vtkCellPicker - pick cells (software ray cast)
- vtkWorldPointPicker - get x-y-z coordinate; does not pick prop (hardware assisted)

---

**Example: Picking and Style**

```cpp
class vtkRenderWindowInteractor*
class vtkInteractorStyleFlight*
class vtkCellPicker*

// Create the render window interactor
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
// Create the interactor style
vtkInteractorStyleFlight *style = vtkInteractorStyleFlight::New();
// Create the cell picker
vtkCellPicker *picker = vtkCellPicker::New();

iren->SetInteractorStyle(flight);
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

- 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

- `vtkObject`
  - `this->InvokeEvent()`
  - Registered observers
    - `vtkCommand`
    - `vtkCommand`
    - `vtkCommand`

3D Widgets

- Added since VTK 4.0 release
  - Requires nightly / VTK 4.2 release
- Subclass of `vtkInteractorObserver`
  - Interactor observers watches 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`
- `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

- Visualization ToolKit - Win32/GTK2 #1
Software Process

- The Quality Dashboard
  - Managing open-source development
  - The importance of feedback

- Tools
  - CVS
  - CMake
  - Doxygen
  - DART

Development Process

- Standard C++ Style / Methodology
- Documentation embedded in code (use Doxygen to generated HTML)
- Use CVS source code control system
  - Allows simultaneous edits
  - Merges files, flags conflicts
- Automatic wrapper generator (look at SWIG)
- Daily regression testing

Insight Toolkit (ITK)

- Cross-Platform Make (build environment)
- Open source, runs everywhere
- Simple “makefiles” CMakeLists.txt
- Generates native build files
  - Makefiles on Unix/Linux
  - Projects/workspaces on MSVC

www.cmake.org
ITK

- **C++ Library**
  - Uses Generic Programming Concepts (i.e. heavily templated)
  - N-Dimensional Images / Meshes
  - Complex Pixel Types
  - BSD-style open-source license
- **Data Flow Architecture**
  - Streaming
- **CABLE Interpreted Language Bindings**
- **Development Process**
  - CMake Build Environment
  - CVS Source Code Control
  - DART Testing System
  - Doxygen Documentation System
  - GNATS Bug Tracking
  - Web (itk.org), Mailing lists (mailman), etc.
- **www.itk.org**