Overview of VTK, pt 2
Announcements

• Quiz #2: Tues Feb 19th. On isolines

• 510 presentations: Tuesday March 10th
  – 2x 5 minute presentations
  • Each 510 student will survey two visualization papers and make a 5 minute presentation on each. Ideally, one paper will be on a visualization application (i.e., how people use visualization) and will include a "movie" that shows the visualizations. The other paper will be on visualization research.
  – Roll will be taken, 2.5 point penalty for missing
CIS 410/510: Project #6
Due Feb 16, 2020
(which means submitted by 6am on Feb 17, 2020)
Worth 8 points

Assignment:
Implement “marching tetrahedrons.” It is similar to the “marching quads” of project 5, but it will require that you extend this algorithm to three dimensions, specifically for tetrahedrons. (Tetrahedrons are the simplest cell type – 4 vertices.)

You will need to make your own conventions for edge numbering, build your own lookup tables, etc.

The skeleton code I provide:
- Has a Tetrahedron class
- Has an empty implementation of “IsosurfaceTet(...)”
- Has the code that loads data and calls IsosurfaceTet

Note this prompt is sparser than previous ones. I figure you all are getting the hang of this by now.

What to upload?:
- Your proj6.cxx file with a working IsosurfaceTet function.
- A screenshot of you running your code and getting the correct output
Final Project: Decision Tree

- This is volume rendering
410 Decision Tree

• Have a custom vis project you want to do?
  – Upload a proposal to Canvas
  – Types of custom project:
    • Movie
      – If you think it sounds fun to make a movie, then pay close attention next two weeks
    • Algorithm, etc.

• If not:
  – Or do volume rendering
510 Decision Tree

• Have a custom vis project you want to do?
  ‒ Upload a proposal to Canvas

• If not:
  ‒ If you think it sounds fun to make a movie, then pay close attention next two weeks

• AND do volume rendering
When are proposals due?

- Sunday Feb 23rd
- By that time you will have seen everything about making movies
- You will not have seen about volume rendering (lectures on Feb 25th & 27th)
- If you see volume rendering lectures and want to switch to a custom project, then that is fine
The Visualization ToolKit

- Open Source library for visualization and graphics
- Written in C++
  - Bindings in Tcl, Java, Python
- Object-oriented
  - Few base types
  - Many, many derived types that provide functionality
- Strengths:
  - Very powerful
  - Designed for extensibility
- Weaknesses:
  - Extensibility design choice comes with performance tradeoffs
Data Flow Overview

• VTK employs the data flow network paradigm

• Basic idea:
  – You have many modules to choose from
    • Hundreds!!
  – You compose some of these modules together to perform some desired functionality

• Advantages:
  – Customizability
  – Design fosters interoperability between modules to the extent possible
Example of data flow (image processing)

- FileReader
- Crop
- Transpose
- Invert
- Color
- Concatenate
- FileWriter
Example of data flow (image processing)

• **Participants:**
  – **Source:** a module that produces data
    • It creates an output
  – **Sink:** a module that consumes data
    • It operates on an input
  – **Filter:** a module that transforms input data to create output data

• **Pipeline:** a collection of sources, filters, and sinks connected together
Key abstract types in VTK

- vtkDataObject / vtkDataSet
- vtkAlgorithm
- Graphics modules
Key abstract types in VTK

• vtkDataObject / vtkDataSet
• vtkAlgorithm
• Graphics modules
Example of data flow (image processing)

We are now talking about the data objects that flow between modules ... not just images.
Key abstract type in VTK: vtkDataObject

While vtkDataObject allows VTK developers to add custom abstractions, almost all usage by new users of VTK is via vtkDataSet.

Key abstract type in VTK: vtkDataObject

I’ve gone 20 years using almost exclusively four concrete types of vtkDataObject.

Important derived types of \texttt{vtkDataSet}

- \texttt{vtkStructuredGrid}
- \texttt{vtkUnstructuredGrid}
- \texttt{vtkRectilinearGrid}
- \texttt{vtkPolyData}
Important methods associated with \texttt{vtkDataSet}

- \texttt{int GetNumberOfCells();}
- \texttt{int GetNumberOfPoints();}
- \texttt{vtkCell *GetCell(int cellID);}
- \texttt{double *GetPoint(int pointID);}
- \texttt{vtkPointData *GetPointData();}
  - Gets fields defined on points (vertices) of mesh
- \texttt{vtkCellData *GetCellData();}
  - Gets fields defined on cells (elements) of mesh
- \texttt{vtkFieldData *GetFieldData();}
  - Gets fields defined not on cells or points

Fields are flexible in VTK, including scalars, vectors, tensors, and fields of arbitrary length.

Polymorphism! ... each derived type implements this interface.

But using this general interface can cost performance. Fixes?
Key abstract types in VTK

- vtkDataObject / vtkDataSet
- vtkAlgorithm
- Graphics modules
Key abstract type in vtk: vtkAlgorithm

• While data flow has clear concepts for “Source”, “Sink”, and “Filter”, VTK has a single class “vtkAlgorithm”
  – Previously had differentiated types
• vtkAlgorithm:
  – has zero, one, or more inputs
    • void SetInputConnection(vtkAlgorithmOutput *); // port 0
    • void SetInputData(int port, vtkAlgorithmOutput *);
  – has zero, one, or more outputs
    • vtkAlgorithmOutput *GetOutputPort(void); // port 0
    • vtkAlgorithmOutput *GetOutput(int);
First program

```c
#include <vtkDataSetReader.h>
#include <vtkContourFilter.h>
#include <vtkDataSetWriter.h>

int main() {
  vtkDataSetReader *rdr = vtkDataSetReader::New();
  rdr->SetFileName("noise.vtk");

  // Contour the data.
  vtkContourFilter *cf = vtkContourFilter::New();
  cf->SetNumberOfContours(1);
  cf->SetValue(0, 3.0);
  cf->SetInputConnection(rdr->GetOutputPort());

  vtkDataSetWriter *wrtr = vtkDataSetWriter::New();
  wrtr->SetFileName("contour.vtk");
  wrtr->SetInputConnection(cf->GetOutputPort());
  wrtr->Write();
}
```

```
fawcett:VTK_ex childs$ make
make -g -o contour_no_graphics -I/Users/childs/visit/vtk/6.0.0/i386-apple-darwin10_gcc-4.2/include/vtk-6.0 contour_no_graphics.C -L/Users/childs/visit/vtk/6.0.0/i386-apple-darwin10_gcc-4.2/lib -lvtkRenderingFreeTypeOpenGL-6.0 -lvtkRenderingFreeType-6.0 -lvtkInteractorStyle-6.0 -lvtkRenderWindow-6.0 -lvtkImage-6.0 -lvtkIOLegacy-6.0 -lvtkIOCore-6.0 -lvtkRenderingCore-6.0 -lvtkFiltersCore-6.0 -lvtkCommonDataModel-6.0 -lvtkCommonMisc-6.0 -lvtkCommonExecutionModel-6.0 -lvtkCommonCore-6.0
fawcett:VTK_ex childs$ ./contour_no_graphics
fawcett:VTK_ex childs$ ls -l contour.vtk
-rw-r--r-- 1 childs staff 1383911 Jul 6 15:47 contour.vtk
```
Modules have many options for how they execute. These options are encoded as attributes in the module and modified using “Setter” functions.
First program

```cpp
fawcett:VTK_ex childs$ cat contour_no_graphics.C
#include <vtkDataSetReader.h>
#include <vtkContourFilter.h>
#include <vtkDataSetWriter.h>

int main()
{
    vtkDataSetReader *rdr = vtkDataSetReader::New();
    rdr->SetFileName("noise.vtk");

    // Contour the data.
    vtkContourFilter *cf = vtkContourFilter::New();
    cf->SetNumberOfContours(1);
    cf->SetValue(0, 3.0);
    cf->SetInputConnection(rdr->GetOutputPort());

    vtkDataSetWriter *wrtr = vtkDataSetWriter::New();
    wrtr->SetFileName("contour.vtk");
    wrtr->SetInputConnection(cf->GetOutputPort());
    wrtr->Write();
}
```

VTK forces all VTK objects to be allocated using dynamic memory (the heap).
VTK memory management

• VTK uses reference counting for all objects (vtkAlgorithm, vtkDataObject, etc)
• Rules:
  – All new objects have a reference count of 1
  – Register() increments the reference count
  – Delete() deletes the reference count
  – When reference count hits 0, the object is deleted
• VTK shares arrays between vtkDataObjects, to save on memory...
  – ... which means they can’t store arrays on stack, since the arrays could go out of scope (dangling pointer)

VTK has recently introduced a templated type, vtkSmartPointer, to assist with reference counting.
First program (leak free version)

```c
#include <vtkDataSetReader.h>
#include <vtkContourFilter.h>
#include <vtkDataSetWriter.h>

int main()
{
    vtkDataSetReader *rdr = vtkDataSetReader::New();
    rdr->SetFileName("noise.vtk");

    // Contour the data.
    vtkContourFilter *cf = vtkContourFilter::New();
    cf->SetNumberOfContours(1);
    cf->SetValue(0, 3.0);
    cf->SetInputConnection(rdr->GetOutputPort());

    vtkDataSetWriter *wrtr = vtkDataSetWriter::New();
    wrtr->SetFileName("contour.vtk");
    wrtr->SetInputConnection(cf->GetOutputPort());
    wrtr->Write();

    rdr->Delete();
    cf->Delete();
    wrtr->Delete();
}
```
First program

```c
#include <vtkDataSetReader.h>
#include <vtkContourFilter.h>
#include <vtkDataSetWriter.h>

int main()
{
  vtkDataSetReader *rdr = vtkDataSetReader::New();
  rdr->SetFileName("noise.vtk");

  // Contour the data.
  vtkContourFilter *cf = vtkContourFilter::New();
  cf->SetNumberOfContours(1);
  cf->SetValue(0, 3, 0);
  cf->SetInputConnection(rdr->GetOutputPort());

  vtkDataSetWriter *wrtr = vtkDataSetWriter::New();
  wrtr->SetFileName("contour.vtk");
  wrtr->SetInputConnection(cf->GetOutputPort());
  wrtr->Write();
}
```

How does VTK control execution?

The pipeline is constructed via `SetInputConnection()` and `GetOutputPort()` calls.
NEW SLIDES
VTK’s Execution Model

• Key method: Update()
  – Update() requests a module to get its output “up-to-date”, i.e., to calculate it

• But what if that modules inputs are not up-to-date?
  – Part of an Update() is to call Update() on all the inputs to a module

• In the example program, “Write()” knows to request its input is up-to-date, which propagates up the pipeline
First program

1) wrtr asks cf to Update()
2) cf asks rdr to Update()
3) rdr reads from the file
4) cf calculates contour
5) wrtr writes file

```c
fawcett:VTK_ex child$ cat contour_no_graphics.C
#include <vtkDataSetReader.h>
#include <vtkContourFilter.h>
#include <vtkDataSetWriter.h>

int main()
{
    vtkDataSetReader *rdr = vtkDataSetReader::New();
    rdr->SetFileName("noise.vtk");

    // Contour the data.
    vtkContourFilter *cf = vtkContourFilter::New();
    cf->SetNumberOfContours(1);
    cf->SetValue(0, 3.0);
    cf->SetInputConnection(rdr->GetOutputPort());

    vtkDataSetWriter *wrtr = vtkDataSetWriter::New();
    wrtr->SetFileName("contour.vtk");
    wrtr->SetInputConnection(cf->GetOutputPort());
    wrtr->Write();
}
```
VTK & Time Stamps

• VTK prevents unnecessarily re-calculation of the pipeline
  – It uses time stamps to keep track of when a module or its input was modified, and when the last time was it calculated its outputed.
First program

```cpp
#include <vtkDataSetReader.h>
#include <vtkContourFilter.h>
#include <vtkDataSetWriter.h>

int main()
{
    vtkDataSetReader *rdr = vtkDataSetReader::New();
    rdr->SetFileName("noise.vtk");

    // Contour the data.
    vtkContourFilter *cf = vtkContourFilter::New();
    cf->SetNumberOfContours(1);
    cf->SetValue(0, 3.0);
    cf->SetInputConnection(rdr->GetOutputPort());

    vtkDataSetWriter *wrtr = vtkDataSetWriter::New();
    wrtr->SetFileName("contour.vtk");
    wrtr->SetInputConnection(cf->GetOutputPort());
    wrtr->Write();

    cf->SetValue(0, 3.5);
    wrtr->SetFileName("contour2.vtk");
    wrtr->Write();

    rdr->Delete();
    cf->Delete();
    wrtr->Delete();
}
```

1) `wrtr` asks `cf` to `Update()`
2) `cf` asks `rdr` to `Update()`
3) `rdr` doesn't read from the file
4) `cf` calculates contour
5) `wrtr` writes file
Topology of pipelines

• Each module can have multiple inputs, multiple outputs
• Multiple sinks are fine
  – Call Update() on each
• Cycles are technically OK, but can be problematic
Key abstract types in VTK

- vtkDataObject / vtkDataSet
- vtkAlgorithm
- Graphics modules
Graphics Modules

• 90+% of VTK source code is sources, sinks, and filters.
• <10% is graphics / windowing.
• ... but ~50% of most “getting started” programs involve graphics / windows
5 Abstractions for Graphics / Windowing

1. **RenderWindow**: a window
2. **Renderer**: the place inside a window where you can render
   - There can be multiple renderers within a window
3. **Actor**: something that can be placed into a renderer
4. **Mapper**: maps data to geometric primitives
   - One mapper can be associated with multiple actors
5. **RenderWindowInteractor**: defines what button clicks, mouse movements, etc. should do
int main()
{
    // The following lines create a sphere represented by polygons.
    //
    vtkSmartPointer<vtkSphereSource> sphere =
        vtkSmartPointer<vtkSphereSource>::New();
    sphere->SetThetaResolution(100);
    sphere->SetPhiResolution(50);
    
    // The mapper is responsible for pushing the geometry into the graphics
    // library. It may also do color mapping, if scalars or other attributes
    // are defined.
    //
    vtkSmartPointer<vtkPolyDataMapper> sphereMapper =
        vtkSmartPointer<vtkPolyDataMapper>::New();
    sphereMapper->SetInputConnection(sphere->GetOutputPort());
    vtkSmartPointer<vtkActor> sphere1 =
        vtkSmartPointer<vtkActor>::New();
    sphere1->SetMapper(sphereMapper);
    sphere1->GetProperty()->SetColor(1,0,0);
    vtkSmartPointer<vtkActor> sphere2 =
        vtkSmartPointer<vtkActor>::New();
    sphere2->SetMapper(sphereMapper);
    sphere2->GetProperty()->SetColor(0,1,0);
    sphere2->AddPosition(1.25,0,0);
}

Adapted from SpecularSpheres.cxx in VTK source code
// Create the graphics structure. The renderer renders into the
// render window. The render window interactor captures mouse events
// and will perform appropriate camera or actor manipulation
// depending on the nature of the events.
//
vtkSmartPointer<vtkRenderer> ren1 =
  vtkSmartPointer<vtkRenderer>::New();
vtkSmartPointer<vtkRenderWindow> renWin =
  vtkSmartPointer<vtkRenderWindow>::New();
renWin->AddRenderer(ren1);
vtkSmartPointer<vtkRenderWindowInteractor> iren =
  vtkSmartPointer<vtkRenderWindowInteractor>::New();
iren->SetRenderWindow(renWin);

// Add the actors to the renderer, set the background and size.
//
ren1->AddActor(sphere1);
ren1->AddActor(sphere2);
ren1->SetBackground(0.1, 0.2, 0.4);
renWin->SetSize(400, 200);

ren1->GetActiveCamera()->SetFocalPoint(0,0,0);
ren1->GetActiveCamera()->SetPosition(0,0,1);
ren1->GetActiveCamera()->SetViewUp(0,1,0);
ren1->GetActiveCamera()->ParallelProjectionOn();
ren1->ResetCamera();
ren1->GetActiveCamera()->SetParallelScale(1.5);

// This starts the event loop and invokes an initial render.
//
iren->Initialize();
iren->Start();

return EXIT_SUCCESS;

More Example Programs

• Many example programs in VTK download
• Some C++, some Python
• Challenge is typically figuring out how to map what you want to do to VTK modules
  – How to find the right module?
  – How to set up the module’s options?
  – Good reference for these questions:
Summary

• VTK is open source, written in C++, and is supported by a large community
• It employs the data flow paradigm
• It has many modules (readers, filters, mappers), which makes it very powerful
• It is well-suited to for many tasks including:
  – foundation for visualization tools
  – one-off visual explorations of data
  – custom visualization tools, especially when considering the effort to incorporate it
Look at 2 movies

David Pugmire

Leigh Orf
• Assume rectilinear mesh with
  – \( X = \{0,1,2,3,4,5,6,7,8,9\} \)
  – \( Y = \{0,1,2,3,4,5,6,7,8,9\} \)
  – \( Z = \{0,1,2,3,4,5,6,7,8,9\} \)

How do we generate slice at \( Y=5 \)?
Slice at Y=5

- **Output mesh:**
  - $X=\{0,1,2,3,4,5,6,7,8,9\}$
  - $Y=\{5\}$
  - $Z=\{0,1,2,3,4,5,6,7,8,9\}$

```c
for (int z = 0 ; z < 10 ; z++)
    for (int x = 0 ; x < 10 ; x++)
        outF[z*10+x] = F[z*100+5*10+x];
```
Slicing

• Assume rectilinear mesh with
  – $X=\{0,1,2,3,4,5,6,7,8,9\}$
  – $Y=\{0,1,2,3,4,5,6,7,8,9\}$
  – $Z=\{0,1,2,3,4,5,6,7,8,9\}$

How do we generate slice at $Y=5.3$?
Slice at Y=5.3

- Output mesh:
  - X={0,1,2,3,4,5,6,7,8,9}
  - Y={5.3}
  - Z={0,1,2,3,4,5,6,7,8,9}

```cpp
for (int z = 0 ; z < 10 ; z++)
    for (int x = 0 ; x < 10 ; x++)
        outF[z*10+x] =
            (0.3)*(F[z*100+6*10+x]-F[z*100+5*10+x])
            + F[z*100+5*10+x];
```
Slicing

- Assume rectilinear mesh with
  - $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $Z = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

How do we generate slice at plane $X+Y+Z=0$?

Answer: we will need “distance functions”
Distance Functions

• Distance function: measures how far a point is from surface

• Example: how far are you from the plane \(X+Y+Z=0\)?

• How far is the point \((3,0,0)\) from this plane?

Answer: 3 units
Distance Functions

• Distance function: measures how far a point is from surface
• Example: how far are you from the plane $X+Y+Z=0$?
• How far is the point $(1,1,0)$ from this plane?

Answer: $\sqrt{2}$ units
How to use distance functions to slice?

- Step #1: create distance function
- Step #2: isosurface with isovalue = 0
Revisiting $Y=5.3$

- Some cells straddle $Y=5.3$
- When classifying, all $Y=5$ get 0, all $Y=6$ get 1
Distance Functions: approximate version

• Distance function: measures how far a point is from surface

• Example: how far are you from the plane $X+Y+Z=0$?

• How far is the point $(x,y,z)$ from this plane?

   Approx. answer: $x+y+z$ units

   Why? Approximate answer: the overestimates (on both sides) cancel each other out.
Threshold

• Keep cell if it meets some criteria, else discard

• Criteria:
  – Pressure > 2
  – 10 < temperature < 20

Cells that meet criteria
How to implement threshold

• Iterate over cells
• If a cell meets the criteria, then place that cell in the output
• Output is an unstructured mesh
Interval Volumes

Isolates portion of volume between two values, $V_{\text{low}}$ and $V_{\text{hi}}$. 
Interval volumes vs isosurfaces

Interval volume between 2.5 and 2.7.

Isosurfaces at 2.5 and 2.7.
How to implement interval volumes

• Iterate over cells
• Like marching cubes, but making topologically 3D output (tetrahedrons, not triangles)
• Now 3 states: below, within interval, above
• Many, many cases to determine
Box

- Isolate portion of volume within a box

- $-8 < x < 8$
- $-9 < y < 5.7$
- $-3.2 < z < 6.4$
How to implement box

- Iterate over cells
- Three cases:
  - Retain cell
  - Discard cell
  - Split cell (i.e., straddles box boundary)
- How to split cell?
  - Box:Interval Volume as Slicing:Isosurfacing
    - (set up 6 distance fields and use interval volumes)
    - (why not 1 distance field?)
Clip by arbitrary functions
How to implement Clip

• Same as Box, but different spatial function
• Iterate over cells
• Three cases:
  – Retain cell
  – Discard cell
  – Split cell (i.e., straddles clip boundary)
• How to split cell?
  – Clip:Interval Volume as Slicing:Isosurfacing
    • (possibly multiple clips)
Slicing by non-planes
How to non-planar slicing

• Set up distance function for spatial function (cone, sphere)
• Apply Marching Cubes
Isosurface by one variable, color by another

Isosurface by var1, color by var1

Isosurface by var1, color by var2
How to implement isosurface by \texttt{var1}, color by \texttt{var2}

- Marching Cubes based on \texttt{var1}.
- Need operation:
  - As Marching Cubes calculates each triangle, evaluate \texttt{var2} for each vertex of that triangle
  - Create variable \texttt{var2} on output triangle mesh