Announcements

• Volume rendering lectures: Friday & next week
  – First: VTK
• Quiz: Nov 9th (isolines)
• Project proposal: due Weds Nov 11th
• Extra OH: Mon 1:30-2:30, Fri 12:30-1:30
• SVN lecture online
• Grading through proj 4 completed
  – Canvas denominators weird
Review
Isosurfacing

We need conventions!
Isosurfacing

We need conventions!
Isosurfacing

We need conventions!
We need conventions!
Isosurfacing

static int edges[12][2] =
  { {0,1}, {1,3}, {2,3}, {0,2},
      {4,5}, {5,7}, {6,7}, {4,6},
      {0,4}, {1,5}, {2,6}, {3,7} };
Isosurfacing

Quiz: where should the isosurface go?
Isosurfacing

Quiz: where should the isosurface go?
**Isosurfacing**

Quiz: where should the isosurface go?
Isosurfacing

Quiz: where should the isosurface go?
Isosurfacing

Quiz: where should the isosurface go?
This is our last two cells, side by side
Uh oh ... ambiguous case is causing problem
Isosurfacing

Uh oh ... ambiguous case is causing problem
Let's try to make something consistent
Isosurfacing

Quiz: how many triangles will we need?

Let’s try to make something consistent
Let's try to make something consistent

No! This is got us into trouble before!
Isosurfacing

Let’s try to make something consistent
Let’s try to make something consistent
Let’s try to make something consistent
From a different angle...
Summary So Far For Ambiguities

• Ambiguities cause problems:
  – If you use one interpretation for one cell, and use the other interpretation for its neighboring cell, then you get gaps
  – Always making the “intuitive” choice does not solve the problem.

• If you choose consistently, then you can avoid these problems.
How to make consistent choices

• “Asymptotic Decider”
  – Analyze scalar field and make decision

Quiz: why does this result in consistent choices?
How to make consistent choices

• Conventions!
  – E.g., always separate lowest vertex
  – This is consistent across faces
  – This is how VTK (and case_checker) works
Project 6B: What should we do?

• Our choices:
  – Figure out those conventions and reproduce them perfectly
    • Correct and additional effort
  – Ignore the conventions and accept gaps
    • Incorrect, but easier

• Note: ambiguous cases don’t come up a lot in practice
Equality

• Current case assignment:
  – $F(V) < \text{isovalue}: 0$
  – $F(V) > \text{isovalue}: 1$

• What if the field value at a vertex is equal to the isovalue?

Quiz: what is the physical interpretation of having $F(v) == \text{isovalue}$?
Equality Strategy

• Case assignment (incorporating inequality):
  – $F(V) < \text{isovalue}: 0$
  – $F(V) \geq \text{isovalue}: 1$
Equality Strategy

• Case assignment (incorporating inequality):
  – \( F(V) < \text{isovalue}: 0 \)
  – \( F(V) \geq \text{isovalue}: 1 \)

• Quiz: calculate isolines for isovalue = 5.
Not Discussed

• Acceleration Structures
  – Will come back to this topic later

• (urgency to prep you all for future HW assignments)
Project 6B

• Bring up prompt and read together
Outline

• Quick introduction to VTK
• Foundational concepts
  – Object-oriented programming
  – Data flow networks
• Overview of key VTK modules
Outline

• Quick introduction to VTK

• Foundational concepts
  – Object-oriented programming
  – Data flow networks

• Overview of key VTK modules
Why VTK is important

• Significant community effort
• Contains hundreds of algorithms
• Used for end user tools...
• ... but also used by computational scientists to do custom visualization
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
VTK: History

• Initially developed in 1993 by three General Electrics researchers
  – Schroeder, Martin, Lorensen
• Kitware Inc. started in 1998 to support VTK
  – world-wide community then begins participating in development
• Usage:
  – basis for many end-user visualization tools
    • VisIt, ParaView, MayaVi, and more ...
  – many, many custom tools
VTK: a mature infrastructure

- Textbook based on VTK, used for many college courses
- Guide devoted to effective usage of VTK software
- Nightly regression tests
- Online documentation
Outline

• Quick introduction to VTK

• Foundational concepts
  – Object-oriented programming
  – Data flow networks

• Overview of key VTK modules
Object-oriented programming (OOP)

• Programming paradigm with “objects”
• Objects contain:
  – data fields
  – methods
• OOP computer programs consist of objects interacting with each other
• C++: class defines the form of an object, and an instance of that class is the object

Hank’s opinion: object-oriented programming is a big deal for VTK because of subtyping and polymorphism
Subtyping

• Subtyping: concept from programming language theory
  – Wikipedia: “a subtype is a datatype that is related to another datatype (the supertype) by some notion of substitutability, meaning that program elements, typically subroutines or functions, written to operate on elements of the supertype can also operate on elements of the subtype”

• “Is a” test
  – If S is a sub-type of T, then S “is a” T
    • Example: T = fruit, S = apple

• Abstract types: define an interface, but no implementation
  This is implemented in C++ using classes, inheritance, and virtual functions.
Polymorphism (generic programming)

- Write code using abstract type
  - Abstract type has methods
  - No usage of concrete types
- Allows for extensibility
  - Can add many new concrete types afterwards
Polymorphism Example

• Abstract type: Shape
• Concrete types: Triangle, Square, Hexagon
• Methods:
  – Shape::GetArea()
  – Shape::GetNumberOfEdges()
• Key points:
  – programs can be written to Shape interface, with no need for knowledge of derived types
  – new concrete types of Shape (e.g., Octagon) can be added afterwards, and not affect existing code

VTK was successful in choosing abstract types that allowed for great extensibility
Also: if you learn abstract types, then you know VTK
Outline

- Quick introduction to VTK
- Foundational concepts
  - Object-oriented programming
  - Data flow networks
- Overview of key VTK modules
Data Flow Overview

• VTK employs the data flow network paradigm

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

• Advantages:
  – Customizability
  – Design fosters interoperability between modules to the extent possible
Data Flow Overview

- **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

- **Nominal inheritance hierarchy:**
  - A filter “is a” source
  - A filter “is a” sink
Example of data flow (image processing)

• Sources:
  – FileReader: reader from file
  – Color: generate image with one color

• Filters:
  – Crop: crop image, leaving only a sub-portion
  – Transpose: view image as a 2D matrix and transpose it
  – Invert: invert colors
  – Concatenate: paste two images together

• Sinks:
  – FileWriter: write to file
Example of data flow (image processing)
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
Benefits of the Data Flow Design

• Extensible!
  – write infrastructure that knows about abstract types (source, sink, filter, and data object)
  – write as many derived types as you want

• Composable!
  – combine filters, sources, and sinks in custom configurations

What do you think the benefits are?
Drawbacks of Data Flow Design

- Operations happen in stages
  - Extra memory needed for intermediate results
  - Not cache efficient
- Compartmentalization can limit possible optimizations
- Abstract interfaces can limit optimizations
Outline

• Quick introduction to VTK
• Foundational concepts
  – Object-oriented programming
  – Data flow networks
• Overview of key VTK modules
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.

I’ve gone 15 years using almost exclusively four concrete types of vtkDataObject.

Important derived types of vtkDataSet

vtkStructuredGrid

vtkUnstructuredGrid

vtkRectilinearGrid

vtkPolyData
Important methods associated with vtkDataSet

- `int GetNumberOfCells();`
- `int GetNumberOfPoints();`
- `vtkCell *GetCell(int cellID);`
- `double *GetPoint(int pointID);`
- `vtkPointData *GetPointData();`
  - Gets fields defined on points (vertices) of mesh
- `vtkCellData *GetCellData();`
  - Gets fields defined on cells (elements) of mesh
- `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?
vtkRectilinearGrid

- Points are implicit
- Cell connectivity is implicit
- Grid of $N_i \times N_j \times N_k$ takes $(N_i + N_j + N_k)$ storage
vtkStructuredGrid

- Also called “curvilinear mesh”
- Points are explicit
- Cell connectivity is implicit
- Grid of Ni x Nj x Nk takes $3 \times (Ni \times Nj \times Nk)$ storage
vtkUnstructuredGrid

- Many supported cell types
  - Tetrahedron, hexahedron, wedge, pyramid, triangle, quadrilateral, higher order, more...
- Points are explicit
- Cells are explicit
- Grid of $N$ points and $M$ cells, with $K$ points per cell takes $3*N + M*K$ storage
vtkPolyData

- Identical to vtkUnstructuredGrid, but cell types are limited to polygonal data
- This is useful for graphics purposes, when rendering surfaces
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$ cat contour_no_graphics.C
fawcett:VTK_ex childs$ make
fawcett:VTK_ex childs$ ls -l contour.vtk
-rw-r--r-- 1 childs staff 1383911 Jul 6 15:47 contour.vtk
```
First program

Modules have many options for how they execute. These options are encoded as attributes in the module and modified using “Setter” functions.

```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();
}
```
First program

```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 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

The pipeline is constructed via SetInputConnection() and GetOutputPort() calls.

How does VTK control execution?

```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’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

```
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();
}
```

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
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

```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());
  rdr->Delete();

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

  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
Example with graphics / windowing (pt 1)

```c++
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, 0.0, 0.0);

    vtkSmartPointer<vtkActor> sphere2 =
    vtkSmartPointer<vtkActor>::New();
    sphere2->SetMapper(sphereMapper);
    sphere2->GetProperty()->SetColor(0.0, 1.0, 0.0);
    sphere2->AddPosition(1.25, 0.0, 0.0);
}
```

Adapted from SpecularSpheres.cxx in VTK source code
Example with graphics / windowing (pt 2)

// 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;

Adapted from SpecularSpheres.cxx in VTK source code
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:
    • http://www.vtk.org/doc/release/6.2/html/
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