CPS 533 Scientific Visualization

Wensheng Shen

Department of Computational Science
SUNY Brockport
Chapter 4: The Visualization Pipeline

This chapter examines the process of data transformation and develops a model of data flow for visualization systems. We will see how the visualization process transforms data into graphics primitives.
4.1 Overview

- Visualization deals with the issues of **transformation** and **representation**.
- Transformation is the process of converting data from its original form into graphics primitives, and eventually into computer images. Example, the process of extracting stock prices and creating an x-y plot depicting stock price as a function of time.
- Representation includes both the internal data structures used to depict the data and the graphics primitives used to display the data. the curve showing stock price change.
- In the example, data structure is two arrays of stock prices and times, x-y plot is the graphics representation.
A data visualization example

\[ F(x, y, z) = a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9 \]

```cpp
#include "vtk.h"

main ()
{
    vtkCamera *camera;
    float range[2];
    vtkRenderer *aren = vtkRenderer::New();
    vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(aren);
    vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWin);
    // Create surface of implicit function
    // Sample quadric function
    vtkQuadric *quadric = vtkQuadric::New();
    quadric->SetCoefficients(1,2,3,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
    vtkSampleFunction *sample = vtkSampleFunction::New();
    sample->SetSampleDimensions(25,25,25);
    sample->SetImplicitFunction(quadric);
    sample->DebugOn();
    // Generate implicit surface
    vtkContourFilter *contour = vtkContourFilter::New();
    contour->SetInput(sample->GetOutput());
    range[0] = 1.0; range[1] = 6.0;
    contour->GenerateValues(3,range);
    contour->DebugOn();
    // Map contour
    vtkPolyDataMapper *contourMapper = vtkPolyDataMapper::New();
    contourMapper->SetInput(contour->GetOutput());
    contourMapper->SetScalarRange(0,7);
    vtkActor *contourActor = vtkActor::New();
    contourActor->SetMapper(contourMapper);
    // Create outline around data
   vtkOutlineFilter *outline = vtkOutlineFilter::New();
    outline->SetInput(sample->GetOutput());
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    vtkPolyDataMapper *outlineMapper = vtkPolyDataMapper::New();
    outlineMapper->SetInput(outline->GetOutput());
    outlineMapper->SetScalarRange(0,7);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    vtkActor *outlineActor = vtkActor::New();
    outlineActor->SetMapper(outlineMapper);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);
    outline->SetOutlineCornerPoints(1,0,0,1,0,0,1,0,0,0,0,0);```

```cpp
vtkLight *light = vtkLight::New();
    light->SetFocalPoint(camera->GetFocalPoint());
    light->SetPosition(camera->GetPosition());
    renWin->AddLight(light);
    renWin->AddActor(contourActor);
    renWin->AddActor(outlineActor);
}

// Clean up
iren->Delete();
aren->Delete();
renWin->Delete();
quadric->Delete();
sample->Delete();
contour->Delete();
contourMapper->Delete();
contourActor->Delete();
outline->Delete();
outlineMapper->Delete();
outlineActor->Delete();
light->Delete();
```
The functional model

Sample $F(x,y,z)$

Point array

Outline

Surface contour

Extract plane

Outline

Lines

Surface contour

Polygons

Extract plane

Points array

Lines

Polygons

Mesh

Line contour

Lines

Display

Display

Display

Display

Oval blocks: operations

Rectangular blocks: data stores (objects)

Arrows: the direction of data movement

Sources: processes with no input data

Sinks: processes with no output data

Filters: processes with both input and output data
The visualization model is the simplified representation of the functional model. In the visualization model, intermediate data stores will not be represented. We assume that they exist as necessary to support the data flow. For example, lines data store is included in the Outline object.
There are two choices in object model:
(1) Combining data stores (object attributes) with processes (methods) into a single object
(2) Separating objects for data stores and processes.

We use the second choice, using one set of objects to represent data, and the other set of objects to represent operations on data, since it is natural to users. In this Case, we need to create a good interface.
4.2 The visualization pipeline

Visualization pipeline

- Data objects
- Process objects
  - Source objects
  - Filter objects
  - Sink objects
    - Mapper objects
    - Writer objects

- Procedural objects
- Reader objects
4.3 Pipeline topology

(a) Single-type system
Input Type = Output Type

(b) Multiple-type system with Enforced type checking

To maintain compatible data type, only compatible types can be connected together in multiple-type systems.
Pipeline connections

(a) Sources, filters, and mappers

- Source: No input, ≥1 output
- Filter: ≥1 input, ≥1 output
- Mapper: ≥1 input, No output

(b) Multiple input and output

Filters:
- Filter: Data

Outputs:
- Data

(a) Sources, filters, and mappers

(b) Multiple input and output
Loops in visualization

Create sample points

Probe data with Points for Velocity $v_i$

Integrate points $X_{i+1} = x_i + v_i \Delta t$

Display points

This example implements linear integration. The sample points are created to initialize The looping process. The output of the integration filter is used to replace the sample Points once the process begins.
4.4 Executing the pipeline

- Two methods of execution: *demand-driven* and *event-driven*.

- In the demand-driven approach, we execute the program only when output is requested, and a large number of changes can be accumulated without intermediate computation. So, this approach minimizes computation.

- In event-driven approach, every change to a process object causes the network to reexecute. So, the output is always up to date.

- There are two ways to synchronize the network execution: explicit and implicit.
Parallel branches need not execute if changes are local to a particular branch
Explicit execution means directly tracking the changes to the network, and then directly control of the execution of process objects. 

**Advantages:** Synchronization is local, we can perform analysis of data flow each time output is requested, and it is suitable for parallel Computing.

**Disadvantages:** each process object depends on the executive, executive cannot easily control execution if the network execution is Conditional.

The explicit approach may be either demand-driven or event-driven.

1. A parameter modified
2. Executive performs dependency analysis
3. Executive executes necessary modules in order of A-B-D-E
Implicit execution

Implicit control means that a process object executes only if its local input or parameter change.
Implicit is implemented by a two-pass process: `Update()` and `Execute()`
Advantages: it is simple and each process needs only know about their direct input.
Disadvantage: it is hard to distribute network execution across computer or to implement complex execution strategies.

1. A parameter modified
2. E output requested
3. Chain E-D-B-A back propagates
   - `Update()` method
4. Chain A-B-D-E execute via
   - `Execute()` method
We may map data through different color lookup tables depending on the variation of range in the data.
4.5 Memory and computation trade-off

Static model serves best when small, variable portions of the network reexecute, and when data sizes are manageable by the computer system. Dynamic model serves the best when the data flows are large, or the same part of the network executes each time. In the dynamic model, each process object releases memory after downstream objects complete execution.
A valuable tool to minimize memory cost is to share storage using *Reference counting*. In reference counting, we allow more than one process object to refer the same data object.

In the example, we have three objects A, B, and C, and assume that each filter A, B, and C shares a common point representation. Other data is local to each object.
4.6 Data interface issues

- Programming interface: the most powerful and flexible approach that allows you to directly program your application to read, write, and process data. This requires the user to learn more about VTK and have the experience on how to construct data objects.

- File interface (readers/writers): readers are source objects and writers are mappers. Example of readers: vtkSTLReader, vtkBYUReader, and writers: vtkSTLWriter, vtkBYUWriter.

- System interface (Importers/Exporters): importers and exporters are objects in the system that read or write data files consisting of more than one objects. Importers and exporters are typically used to restore or save entire scene, such as lights, cameras, actors, data, transformations, etc. Examples of importers and exporters are vtk3DSIImporter and vtkVRMLExporter.
Importing and exporting files in vtk

// import from 3d studio
vtk3DImport *importer = vtk3DImport::New();
importer -> ComputeNormalsOn();
importer -> SetFileName("../../../vtkdata/Viewport/iflamigm.3ds");
importer -> Read();

// export to rib format
vtkRIBExporter *exporter = vtkRIBExporter::New();
exporter -> SetFilePrefix(importExport);
exporter -> SetRenderWindow(importer->GetRenderWindow());
exporter -> Write();
Data flow in vtk

Description of implicit execution process implemented in vtk. The Update() method is initiated via the Render() method from the actor. Data flows back to mapper via Execute() method. Arrows connecting objects indicate direction of Update() process.
vtkSphereSource *sphere = vtkSphereSource::New();
sphere->SetPhiResolution(12);
sphere->SetThetaResolution(12);

tvtkElevationFilter *colorIt = vtkElevationFilter::New();
colorIt->SetInput(sphere->GetOutput());
colorIt->SetLowPoint(0,0,-1);
colorIt->SetHighPoint(0,0,1);

tvtkDataSetMapper *mapper = vtkDataSetMapper::New();
mapper->SetInput(colorIt->GetOutput());

vtkActor *actor = vtkActor::New();
actor->SetMapper(mapper);
vtkSphereSource *sphere = vtkSphereSource::New();
sphere->SetPhiResolution(12);
sphere->SetThetaResolution(12);

vtkTransform *aTransform = vtkTransform::New();
aTransform->Scale(1,1.5,2);

vtkTransformFilter *transFilter = vtkTransformFilter::New();
transFilter->SetInput(sphere->GetOutput());
transFilter->SetTransform(aTransform);

vtkElevationFilter *colorIt = vtkElevationFilter::New();
colorIt->SetInput(sphere->GetOutput());
colorIt->SetLowPoint(0,0,-1);
colorIt->SetHighPoint(0,0,1);

vtkLookupTable *lut = vtkLookupTable::New();
lut->SetHueRange(0,0);
lut->SetSaturationRange(0,0);
lut->SetValueRange(0.1,1);

vtkDataSetMapper *mapper = vtkDataSetMapper::New();
mapper->SetInput(colorIt->GetOutput());

vtkActor *actor = vtkActor::New();
actor->SetMapper(mapper);
vtkSphereSource *sphere = vtkSphereSource::New();
sphere->SetPhiResolution(12);
sphere->SetThetaResolution(12);

vtkPolyDataMapper *sphereMapper = vtkPolyDataMapper::New();
sphereMapper->SetInput(sphere->GetOutput());

vtkActor *sphereActor = vtkActor::New();
sphereActor->SetMapper(sphereMapper);

vtkConeSource *cone = vtkConeSource::New();
cone->SetResolution(6);

vtkGlyph3D *glyph = vtkGlyph3D::New();
glyph->SetInput(sphere->GetOutput());
glyph->SetSource(cone->GetOutput());
glyph->SetVectorModeToUseNormal();
glyph->SetScaleModeToScaleByVector();
glyph->SetScaleFactor(0.25);

vtkPolyDataMapper *spikeMapper = vtkPolyDataMapper::New();
spikeMapper->SetInput(glyph->GetOutput());

vtkActor *spikeActor = vtkActor::New();
spikeActor->SetMapper(spikeMapper);
vtkSphereSource *sphere = vtkSphereSource::New();
sphere->SetPhiResolution(12);
sphere->SetThetaResolution(12);

vtkShrinkFilter *shrink = vtkShrinkFilter::New();
shrink->SetInput(sphere->GetOutput());
shrink->SetShrinkFactor(0.9);

vtkElevationFilter *colorIt = vtkElevationFilter::New();
colorIt->SetInput(shrink->GetOutput());
colorIt->SetLowPoint(0,0-0.5);
colorIt->SetHighPoint(0,0,0.5);

vtkDataSetMapper *mapper = vtkDataSetMapper::New();
mapper->SetInput(colorIt->GetOutput());

vtkActor *actor = vtkActor::New();
actor->SetMapper(mapper);

renWin->Render(); //execute first time
Shrink->SetInput(colorIt->GetOutput()); //create loop
renWin->Render(); //begin looping;