Interpolation

2D
Given:

Needed:

1D
Given:

Needed:

General Process

Original function
Acquisition
Reconstructed Function
Reconstruction
Sampled function
Resampling
Re-sampled function
Interpolation functions

In general, if \( P \) falls into a cell:
\[
\mathbf{C} = \{P_0, P_1, \ldots, P_{k-1}\}
\]
the attribute of \( P \) is computed by a weighted average of \( P_i \) (\( i=0 \ldots k-1 \))
\[
f(P) = \sum_{i=0}^{k-1} w_i \cdot f(P_i)
\]
where
\[
\sum_{i=0}^{k-1} w_i = 1, \ w_i > 0
\]

Interpolation with cells

- **Line segment**
  \[
d(t) = (1-t)d_0 + t \cdot d_1
\]
- **Pixel and quadrilateral**
  \[
  d = (1-v) \cdot D_0 + v \cdot D_1 \\
  D_0 = (1-u) \cdot d_0 + u \cdot d_1 \\
  D_1 = (1-u) \cdot d_2 + u \cdot d_3
\]
- **Polygon:**
  \[
  d = \sum_{i=0}^{n-1} \left(1/r_i^2\right) \cdot d_i \left/ \sum_{i=0}^{n-1} (1/r_i)^2 \right. \\
  \text{(where } r_i = |p - p_i|)\]
Triangle: Barycentric Coordinates

\[ d = u \cdot d_0 + v \cdot d_1 + w \cdot d_2 \]

\[ N = \text{Normalize}(\text{Cross}(B-A,C-A)); \]
\[ \text{AreaABC} = \text{Dot}(N, \text{Cross}(B-A,C-A)); \]
\[ \text{AreaPBC} = \text{Dot}(N, \text{Cross}(B-P,C-P)); \]
\[ u = \text{AreaPBC} / \text{AreaABC}; \]
\[ \text{AreaPCA} = \text{Dot}(N, \text{Cross}(C-P,A-P)); \]
\[ v = \text{AreaPCA} / \text{AreaABC}; \]
\[ w = 1 - u - v; \]

Interpolation with cells (3D)

✓ Tetrahedron:

\[ d = u \cdot d_0 + v \cdot d_1 + w \cdot d_2 + (1 - u - v - w) \cdot d_3 \]

✓ Voxel

\[ \begin{align*}
    d &= (1-u) \cdot T_0 + u \cdot T_1 \\
    T_0 &= (1-v) \cdot D_0 + v \cdot D_2 \\
    T_1 &= (1-v) \cdot D_1 + v \cdot D_3 \\
    D_0 &= (1-w) \cdot d_0 + w \cdot d_4 \\
    D_1 &= (1-w) \cdot d_1 + w \cdot d_8 \\
    D_2 &= (1-w) \cdot d_2 + w \cdot d_6 \\
    D_3 &= (1-w) \cdot d_3 + w \cdot d_7
\end{align*} \]
Derivative computation

✓ Computing the derivatives of the field at sampling points
✓ Necessary for shading and many other operations
✓ Approximated from sampling points
✓ LCS to WCS conversion

Derivative computation (2)

Gradient computation for volume data:

\[
\text{grad}(f) = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right]
\]

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial x} + \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial x} + \frac{\partial f}{\partial w} \cdot \frac{\partial w}{\partial x}
\]

\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial y} + \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial y} + \frac{\partial f}{\partial w} \cdot \frac{\partial w}{\partial y}
\]

\[
\frac{\partial f}{\partial z} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial z} + \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial z} + \frac{\partial f}{\partial w} \cdot \frac{\partial w}{\partial z}
\]

From interpolation function:

\[
\frac{\partial f}{\partial u} = T_1 - T_0
\]

\[
\frac{\partial f}{\partial v} = (1-v) \cdot (1-u) \cdot d_0 + u \cdot D_0 - (1-u) \cdot D_0 + u \cdot D_1
\]

\[
\frac{\partial f}{\partial w} = [(1-v) \cdot ((1-u) \cdot d_4 + u \cdot d_3) + v \cdot ((1-u) \cdot d_6 + u \cdot d_7)]
\]

\[
-[(1-v) \cdot ((1-u) \cdot d_0 + u \cdot d_1) + v \cdot ((1-u) \cdot d_2 + u \cdot d_3)]
\]
\[ F(u) = \int f(x)[\cos(2\pi ux) - i\sin(2\pi ux)]dx = R(u) + iI(u) \]

\[ f(x) = \int F(u)[\cos(2\pi ux) + i\sin(2\pi ux)]du \]
Fig. 14.15 Signals in the spatial and frequency domains. (a) Sine. (b) Square Wave. (c) Monotone. The DC value in the frequency domain is truncated to make the other values legible and should be 128. (Courtesy of George Wolberg, Columbia University.)
\( f_h \): Highest frequency
If \( f_h < \infty \): Band limited
If \( f_h = \infty \): Band Unlimited
\[ r_n = 2f_n \]: Nyquist rate; In general: \( r > r_n \)

Fig. 14.16 Sampling at the Nyquist rate (a) at peaks, (b) between peaks, (c) at zero crossings. (Courtesy of George Wolberg, Columbia University.)

Fig. 14.17 Sampling below the Nyquist rate. (Courtesy of George Wolberg, Columbia University.)
Fig. 14.20 The sampling process with filtering. (Courtesy of George Wallberg, Columbia University.)

Fig. 14.21 Low-pass filtering in the frequency domain. (a) Original spectrum. (b) Low-pass filter. (c) Spectrum with filter. (d) Filtered spectrum. (Courtesy of George Wallberg, Columbia University.)
**Convolution:**

\[ h(x) = f(x) * g(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau. \]

- Pulse in one domain ⇔ Sinc in another domain
- Multiplication in frequency domain ⇔ Convolution in spatial domain
- Support: non-zero interval of the convolution function (filter, kernel)
Fig. 14.23 Low-pass filtering in the spatial domain. (a) Original signal. (b) Sinc filter. (c) Signal with filter, with value of filtered signal shown as a black dot (x) at filter's origin. (d) Filtered signal. (Courtesy of George Wolfberg, Columbia University.)

Fig. 14.26 (a) Comb function and (b) its Fourier transform. Convoluting the comb's Fourier transform with (c) a signal's Fourier transform in the frequency domain yields (d) the replicated spectrum of the sampled signal. (Courtesy of George Wolfberg, Columbia University.)
Reconstruction Kernels

- Nearest Neighbor (Box)
- Linear
- Sinc
- Gaussian
- + many others

Spatial d. | Frequency d.
---------|---------

Sources of Aliasing

- Non-bandlimited signal
- Low sampling rate (below Nyquist)
- Non perfect reconstruction
Higher Dimensions

- An-isotropic Filters
  - (radially symmetric)
  \[ h(x, y) = h(\sqrt{x^2 + y^2}) \]
- separable filters
  \[ h(x, y) = h(x) \cdot h(y) \]

Interpolation (summary)

- Very important; regardless of algorithm
- expensive => done very often for one image
- Requirements for good reconstruction
  - performance
  - stability of the numerical algorithm
  - accuracy

Nearest neighbor

Linear
VTK: A Scientific Visualization and Graphics System


✓ VTK Software
  http://www.vtk.org/

---

VTK as a toolkit

✓ C++ Core
  - Each module is a C++ class
  - Type of connections enforced by compiler
  - Connect by *SetInput()* and *GetOutput()* methods

✓ Portable across platforms (e.g. built on openGL)
✓ Interface support through Tcl/Tk and Java interpreters
VTK Graphics Model

- Scene consists of objects, called *actors* in VTK, viewed by virtual camera.
- *vtkActor*
  - Has polygon geometry
  - Positions and geometry in world coordinate system
- *Camera* and *Lights* can be defined (there are default light and camera parameters)
- Actors and camera can be transformed
- Actor transformation handled by *vtkProp* (a super class of *vtkActor*)

VTK graphics model (2)

- **Windows:** *vtkRenderWindow*
  - Can have multiple windows (instances)
- **Renderer:** *vtkRenderer*
  - Coordinates rendering process, handle lights, camera and objects.
  - Can have multiple renderers in a window, each has its own viewports, objects, mappers and rendering properties.
VTK Graphics Model (3)

- Props (vtkProp):
  - super class of vtkActor, vtkDataset, vtkVolume, etc.
  - objects added to a rendered to create a scene
  - associated with a mapper and a property object
  - Actor: representing 3D geometric objects

- Mapper: vtkMapper, vtkDataSetMapper, vtkPolyDataMapper, etc.
  - Rendering methods for data and objects

- Properties: vtkProperty, vtkVolumeProperty, etc.
  - Rendering parameters
Interact with rendering window

✓ Interactor: `vtkRenderWindowInteractor`
   - Attached to a window
   - Activating interactor for event-handling
   - Event-driven pipeline execution

✓ Interactor functions:
   - `Rotate` (left mouse button)
   - `Zoom` (right mouse button)
   - `Pan` (left mouse+shift)
   - “w”: draw wireframe
   - “s”: draw surface mesh
   - “r”: reset camera
   - “e”: exit
   - “p”: pick actor (underneath mouse pointer)

Example

```cpp
vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->SetSize(600,300);
vtkRenderer *ren1 = vtkRenderer::New();
vtkRenderer *ren2 = vtkRenderer::New();
ren1->SetViewport(0.0,0.0,0.5,1.0);
ren1->SetBackground(0.8,0.4,0.2);
ren2->SetViewport(0.5,0.0,1.0,1.0);
ren2->SetBackground(0.1,0.2,0.4);
renWin->AddRenderer(ren1);
renWin->AddRenderer(ren2);
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);
iren->Render();
iren->Start();
```
Reader, Writer, Source, Filter, etc

- **Reader:** `vtkDataSetReader`, `vtkPolyDataReader`, `vtkVolumeReader`, `vtkStructuredGridReader`, `vtkPNMReader`, etc.
  - Reading from datafile
- **Writer:** `vtkDataSetWriter`, `vtkPolyDataWriter`, `vtkUnstructuredGridWriter`, `vtkPNMWriter`, etc.
  - Write to datafile
- **Source:** `vtkCubeSource`, `vtkStructuredGridSource`, etc
  - Source object
- **Filter:** `vtkContourFilter`, `vtkStreamLine`, `vtkHedgehog`, ...
  - Filter object
- **Misc:** `vtkCamera`, `vtkLight`, etc.

Example: rendering geometric objects

```c++
vtkCubeSource *cube = vtkCubeSource::new();
vtkPolyDataMapper *mapper = vtkPolyDataMapper::new();
mapper->SetInput(cube->GetOutput());

vtkActor *actor = vtkActor::new();
actor->SetMapper(mapper);
ren1->AddProp(actor);
actor->RotateX(30.0); actor->RotateY(20.0);
actor->GetProperty()->SetColor(1.0,.07,.07);
ren1->ResetCamera();
renWin->Render();
```
The VTK Model - a visualization pipeline

Data Object → Process Object → Display

Source: Procedural, Reader
Filter: Transforms the data
Mapper: Creates Graphics Primitives

Data Representation: Cells & Points

- **Topology**
  - Shape such as triangle, tetrahedron
- **Geometry**
  - Point Coordinates assigned to a topology
- **Data Attributes**
  - Data associated with topology or geometry
Cells specify Topology

- Vertex
- Polyvertex
- Line
- Polyline
- Triangle
- Triangle Strip
- Quadrilateral
- Polygon
- Tetrahedron
- Hexahedron
- Voxel

Cells

- Cell is defined by an ordered list of points
  - Triangle, quadrilateral points specified counter clockwise
  - Others as shown
Meshes are made of Cells

- Cells can be many different shapes and sizes
  - 2D: Triangles, Quadrilaterals, etc
  - 3D: Tetrahedra, Hexahedra, Pyramids, etc.
- Meshes can consist of one or more types of cells

VTK Dataset Types

- `vtkStructuredPoints`
- `vtkRectilinearGrid`
- `vtkStructuredGrid`
- `vtkPolyData`
- `vtkUnstructuredGrid`
- Methods for reading and writing
Datasets

Organizing structure plus attributes
Structured points

- Rectilinear Grid
- Structured Grid

Unstructured Grid

A collection of vertices, edges, faces and cells whose connectivity information must be explicitly stored
Data Attributes Assigned to points (VTK) or cells

- Scalars
- Vector
  - Magnitude and direction
- Normal
  - A vector of magnitude 1
  - Used for lighting
- Texture Coordinate
  - Mapping data points into a texture space
- Tensor

Visualization of Attributes

- Scalar
  - Color Mapping
  - Contouring
    - 3D Isosurface

Contour Value of 5
Visualization of Attributes

- Vectors
  - Oriented Lines
  - Oriented Glyphs
  - Streamlines

Visualization Pipeline

Source → Filter → Mapper → Actor

Data → Filter → Mapper → Actor

Data → Filter → Actor

to graphics system
Datasets

- vtkDataSet
  - vtkStructuredPoints
    - implicit topology/geometry
  - vtkPointSet
    - explicit geometry
  - vtkRectilinearGrid
    - implicit topology, semi-explicit geometry
  - vtkStructuredGrid
    - implicit topology
  - vtkPolyData
  - vtkUnstructuredGrid

Data Objects

- vtkPolyData
  - vtkStructuredPoints
  - vtkRectilinearGrid
  - vtkUnstructuredGrid
  - vtkStructuredGrid
Process Objects

Source

1 or more inputs

1 or more outputs

Filter

1 or more inputs

1 or more outputs

Mapper

Creating The 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
VTK File Format

- Several standard file formats supported
- **VTK file format:**
  1. Title line:
  2. Header line:
  3. Format line:
  4. Dataset structure:
  5. Dataset attributes:

```
# vtk DataFile Version 2.3
This is a sample datafile
ASCII (or BINARY)
DATASET type
......
POINT_DATA n
......
CELL_DATA n
......
```

VTK Examples

http://www.vtk.org/Wiki/VTK/Examples/Cxx
A VTK PROGRAM

main()
{
    vtkRenderer *renderer = vtkRenderer::New();
    vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(renderer);
    vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWin);
    vtkSphereSource *sphere = vtkSphereSource::New();
    sphere->SetPhiResolution(12); sphere->SetThetaResolution(12);
    vtkElevationFilter *colorIt = vtkElevationFilter::New();
    colorIt->SetInput(sphere->GetOutput());
    colorIt->SetLowPoint(0,0,-1);
    colorIt->SetHighPoint(0,0,1);
    vtkDataSetMapper *mapper = vtkDataSetMapper::New();
    mapper->SetInput(colorIt->GetOutput());

    vtkActor *actor = vtkActor::New();
    actor->SetMapper(mapper);
    renderer->AddActor(actor);
    renderer->SetBackground(1,1,1);
    renWin->SetSize(450,450);
    renWin->Render();
    iren->Start();
    // Clean up
    renderer->Delete();
    renWin->Delete();
    iren->Delete();
    sphere->Delete();
    colorIt->Delete();
    mapper->Delete();
    actor->Delete();
}
Rendering Volume Data

```c
vtkSLCReader *vReader = vtkSLCReader::new();
vReader->SetFileName("hip.slc");
vtkVolumeTextureMapper2D *vMapper = vtkVolumeTextureMapper2D::new();
vMapper->SetInput(vReader->getOutput());
vtkPiecewiseFunction *vOpacity = vtkPiecewiseFunction::new();
vOpacity->AddPoint(0,0.0); vOpacity->AddPoint(255,0.2);
vtkColorTransferFunction *vColor = vtkColorTransferFunction::new();
vColor->AddRGBPoint(64,1.0,0.0,0.0);
vColor->AddRGBPoint(128,0.0,0.0,1.0);
vColor->AddRGBPoint(196,0.0,1.0,0.0);
vtkVolumeProperty *vProp = vtkVolumeProperty::new();
vProp->SetColor(vColor);
vProp->SetScalarOpacity(vOpacity);
```
vtkVolume *volume = vtkVolume::new();
volume->SetMapper(vMapper);
volume->SetProperty(vProp);
ren2->AddProp(volume);

vtkPolyDataReader *sReader = vtkPolyDataReader::new();
sReader->SetFileName("hipSurface.vtk");
vtkPolyDataMapper *sMapper = vtkPolyDataMapper::new();
sMapper->SetInput(sReader->GetOutput());
vtkActor *sActor = vtkActor::new();
sActor->SetMapper(sMapper);
Ren2->AddProp(sActor);
ren2->ResetCamera();
renWin->Render();