CS53000 - Spring 2018

Introduction to Scientific Visualization

Grids and Data Reconstruction

January 16, 2018
Outline

• Mesh types
• Interpolation
• Spatial queries
• Common pre-processing tasks
Input Data

- Discrete positions (called *vertices*)
- \( N \) dimensions, \( N=1, 2, 3, \ldots \)
- With or without *connectivity information*
  - Structured
  - Unstructured
  - Scattered
Input Data

- Discrete positions (called *vertices*)
- $N$ dimensions, $N=1, 2, 3, ...$
- With or without *connectivity information*
  - Structured
  - Unstructured
  - Scattered

grid topology
Mesh Classification

- **Geometry**
  - Position of vertices in space
  - Structured (follows a pattern) / unstructured

- **Topology**
  - Cells
  - Connectivity information
  - Neighborhood definition
  - Structured / unstructured
Mesh Geometry

Uniform

- implicit relationship between points
- positions can be computed (procedural)

\[ P_{i,j,k} = P_{0,0} + i\Delta_x \vec{e}_x + j\Delta_y \vec{e}_y + k\Delta_z \vec{e}_z \]
Mesh Geometry

Structured

• implicit relationship between points
• positions can be computed (procedural)

\[ P_{i,j,k} = P_{0,0} + \Delta_x[i] \vec{e}_x + \Delta_y[j] \vec{e}_y + \Delta_z[k] \vec{e}_z \]
Mesh Geometry

Unstructured

- No underlying structure
- Requires explicit knowledge of every vertex’s position: \((x_0, y_0, z_0), (x_1, y_1, z_1), \ldots,\)
Mesh Topology

**Structured** *(cf. quadrilateral / hexahedron)*

- Implicit connectivity between vertices
- Implicit cell definition

When associated with **structured geometry**: very efficient point location
Mesh Topology

Structured \((\text{cf. quadrilateral} / \text{hexahedron})\)

- Implicit connectivity between vertices
- Implicit cell definition

When associated with structured geometry:

- very efficient position location
- implicit triangulation
Mesh Topology

Structured (quadrilateral / hexahedron)

- Implicit connectivity between vertices
- Implicit cell definition

When associated with unstructured geometry:

curvilinear grid
Mesh Topology

Structured \((\text{quadrilateral} / \text{hexahedron})\)

- Implicit connectivity between vertices
- Implicit cell definition
- \textbf{Uniform grid} in computational space

\[ \begin{array}{ccccccccc}
\vdots \\
(n-2)m & \ldots & (n-1)m \\
2m & m & \ldots & 1 & 0 \\
m & m & \ldots & m-1 & m-2 \\
\end{array} \]

\begin{array}{ccccccccc}
\text{Physical Space} \\
\text{Computational Space} \\
\end{array}
Mesh Topology

Unstructured *(any cell type)*

- Explicit cell definition
- Types
- Vertices

- Tetrahedron
- Hexahedron
- Wedge
- Pyramid
## Mesh Types Summary

<table>
<thead>
<tr>
<th>Geometry</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>Image</td>
</tr>
<tr>
<td>Structured</td>
<td>Rectilinear</td>
</tr>
<tr>
<td>Unstructured</td>
<td>Curvilinear</td>
</tr>
</tbody>
</table>
Mesh Topology

- **Mesh-free** (no connectivity)
Outline

• Mesh types
• Interpolation
• Spatial queries
• Common pre-processing tasks
Interpolation

• **Continuous reconstruction of discrete input data**

\[ F : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[(x_i, f_i) \]

\[ \forall i \in \{1,..,n\}, F(x_i) = f_i \]

• Data is discrete but it represents a continuous phenomenon.

• Depends on mesh structure (if available)

• Interpolation vs. approximation
Linear Interpolation

- In triangle (2D simplex)
  \[ \phi(x, y) = a + bx + cy \]
  \[ \forall i, \phi(P_i) = \phi_i \]
Linear Interpolation

- **Barycentric coordinates**

\[
P = \sum_{i=1}^{3} \beta_i P_i \]

\[
\sum_{i=1}^{3} \beta_i = 1
\]

\[
\phi(P) = \beta_1(P)\phi_1 + \beta_2(P)\phi_2 + \beta_3(P)\phi_3
\]
Linear Interpolation

- **Barycentric coordinates**
  - $P$ lies outside triangle $\iff$ at least one $\beta_i$ is negative
  - Easy way to check if a position lies inside a triangle
Linear Interpolation

• In tetrahedron (3D simplex)
  • Barycentric coordinates
    \[ P = \sum_{i=1}^{4} \beta_i P_i, \quad \sum_{i=1}^{4} \beta_i = 1 \]
    \[ \phi(P) = \sum_{i=1}^{4} \beta_i(P) \phi_i \]
  • Same inclusion test
Bilinear Interpolation

• In quadrilateral cell

\[ \phi(x, y) = axy + bx + cy + d \]

\[ \forall i, \phi(P_i) = \phi_i \]

Combination of two consecutive linear interpolations
Bilinear Interpolation

- In rectangle

\[
P = (1 - v)Q_b(u) + vQ_t(u) = (1 - u)R_l(v) + uR_r(v)
\]
Bilinear Interpolation

- In rectangle

\[ P = P_1 + u(P_2 - P_1) + v(P_4 - P_1) + uv(P_1 - P_2 + P_3 - P_4) \]
Bilinear Interpolation

• In rectangle

\[ \phi(P) = \phi_1 + u(\phi_2 - \phi_1) + v(\phi_4 - \phi_1) + uv(\phi_1 - \phi_2 + \phi_3 - \phi_4) \]
Bilinear Interpolation

- In arbitrary quadrilateral

  i. If we know coordinates \((u,v)\) in computational space: apply previous formula to obtain position in physical space (easy)
Bilinear Interpolation

- In arbitrary quadrilateral
  
  i. If we know coordinates \((u,v)\) in computational space: apply previous formula to obtain position in physical space (easy)

Important: axis-parallel straight lines are mapped to straight lines
Bilinear Interpolation

- In arbitrary quadrilateral
  
  ii. If we only know coordinates (x,y) in physical space
    
    1. Compute (inverse) mapping from physical space to computational space (unit square): quadratic equations
    2. Apply bilinear formula
Trilinear Interpolation

- **In a cuboid** (axis parallel)
  - general polynomial expression
    \[ \phi(x, y, z) = axyz + bxy + cxz + dyz + ex + fy + gz + h \]
  - expressed in local coordinates
    \[
    P = P_1 + u(P_2 - P_1) + \\
v(P_4 - P_1) + w(P_5 - P_1) + \\
uv(P_1 - P_2 + P_3 - P_4) + \\
w(P_1 - P_2 - P_5 + P_6) + \\
vw(P_1 - P_4 - P_5 + P_8) + \\
uvw(-P_1 + P_2 - P_3 + P_4 - P_5 + P_6 - P_7 + P_8)
    \]
Trilinear Interpolation

- In arbitrary *hexahedron*
  
i. If we know coordinates \((u,v,w)\) in computational space, apply previous formula *(easy)*
Trilinear Interpolation

• In arbitrary hexahedron

ii. If we only know coordinates \((x,y,z)\) in physical space:
   
   1. compute inverse mapping from physical space to computational space (axis aligned unit cube)
      
      i. nonlinear problem

      \[
      (x, y, z)^T = F(u, v, w)^T
      \]

      ii. solved numerically with iterative scheme

      \[
      F(u + \delta) = F(u) + J\delta + ... \]

      \[
      J\delta = -F(u)
      \]

2. goto i.
Trilinear Interpolation

- In **pyramids** (special case of trilinear int.)

\[ P(u, v, w) = (1 - u)(1 - v)(1 - w)P_0 + u(1 - v)(1 - w)P_1 + uv(1 - w)P_2 + (1 - u)v(1 - w)P_3 + wP_4 \]

*Note: Set \( P_5 = P_6 = P_7 = P_4 \) in formula on s. 31*
Trilinear Interpolation

- In prisms (special case of trilinear int.)

\[ P(u, v, w) = (1 - u)(1 - v)(1 - w)P_0 + u(1 - v)(1 - w)P_1 + uv(1 - w)P_2 + (1 - u)v(1 - w)P_3 + (1 - u)wP_4 + uwP_5 \]

Note: Set \( P_6 := P_5 \) and \( P_7 := P_4 \) in formula on s. 31
Scattered Data Interpolation

- Shepard methods

\[
\sigma_i(x) = \frac{1}{||x - x_i||^k}
\]

Original Shepard
- flat spots
- global
Scattered Data Interpolation

• Shepard methods

http://diwww.epfl.ch/w3lsp/publications/other/sdimfeisas.pdf
Scattered Data Interpolation

- **Shepard methods**

\[
F(x) = \sum_{i=1}^{N} \omega_i(x) f_i
\]

\[
\omega_i(x) = \frac{\sigma_i(x)}{\sum_{j=1}^{N} \sigma_j(x)}
\]

\[
\sigma_i(x) = \exp \left(-\left(\alpha_i(x - x_i)^2 + \beta_i(y - y_i)^2 + \gamma_i(z - z_i)^2\right)\right)
\]
Scattered Data Interpolation

• Shepard methods

\[
F(x) = \sum_{i=1}^{N} \omega_i(x) Q_i(x)
\]

\[
\omega_i(x) = \frac{\sigma_i(x)}{\sum_{j=1}^{N} \sigma_j(x)}
\]

\[
\sigma_i(x) = \frac{1}{\|x - x_i\|} \left(1 - \frac{\|x - x_i\|}{R_\omega}\right)^2 +
\]

Franke, Nielson quadratic fit local
Radial Basis Functions

- Set of points: \( X = (x_i)_{i=1}^N \subset \mathbb{R}^d \)
- Associated values: \( (f_i)_{i=1}^N \subset \mathbb{R} \)
- Find coefficients \( (c_i)_{i=1}^N \subset \mathbb{R} \)
  satisfying
\[
  P_f(x) = \sum_{i=1}^N c_i \phi(||x - x_i||)
\]
- System of equations: \( A \mathbf{c} = \mathbf{f} \)
  with \( A_{ij} = \phi(||x_i - x_j||) \)
Radial Basis Functions

• Hardy’s inverse multiquadrics
  \[ f(r) = \frac{1}{\sqrt{r^2 + c}} \]

• Other radial functions used in practice

  • Thin plate splines
    \[ f(r) = r^2 \ln r \]

  • Truncated gaussians
    \[ f(r) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{r^2}{2\sigma^2}} \]

  • Polyharmonic splines
    \[ f(r) = r^\beta \]
But Also...

- Nearest Neighbor interpolation

Voronoi diagram
But Also...

• Higher-order interpolation schemes
  - splines, local polynomial fit (interpolation, least squares, ...)
  - smooth reconstruction kernels (on uniform grids)
Outline

• Mesh types
• Interpolation
• Spatial queries
• Common pre-processing tasks
Context

- Queries in large, unstructured grids
  - arbitrary probing and interpolation
  - resampling (e.g., on regular grid)
- Goal: speed up computation
- Expensive: avoid it when you can
  - iteration (loop over grid vertices)
  - use neighborhood information: leverage correlation between consecutive queries (e.g., streamline, isosurface)
Neighborhood Data

• Data structures store top-down relationships (cell to vertex)

• No direct support for neighborhood information (vertex to cell, cell to cell)
Neighborhood Data

- Bottom-up relationships obtained by computation

- Cell to cell relationship: cell → vertices → cells
Regular Space Subdivision

- Overlay uniform/rectilinear grid over unstructured domain
  - store in each bucket the IDs of intersected cells (bbox test)
Regular Space Subdivision

- Overlay uniform/rectilinear grid over unstructured domain
  - store in each bucket the IDs of intersected cells (bbox test)
Regular Space Subdivision

• Overlay uniform/rectilinear grid over unstructured domain
  • store in each bucket the IDs of intersected cells (bbox test)
Regular Space Subdivision

- Overlay uniform/rectilinear grid over unstructured domain
  - store in each bucket the IDs of intersected cells (bbox test)
Regular Space Subdivision

✓ Easy to implement
✓ Works well if cell distribution is fairly uniform
  - Extremely slow for highly unstructured meshes
    ➡ many cells per bucket
    ➡ many buckets per cell
“Jump & Walk” Approach

• Use data structure to quickly determine “good” starting point
• Using cell information, walk across cells from starting point until cell containing target point is reached.
“Jump & Walk” Approach

• Use data structure to quickly determine “good” starting point

• Using cell information, walk across cells from starting point until cell containing target point is reached.
“Jump & Walk” Approach

- Use data structure to quickly determine “good” starting point
- Using cell information, walk across cells from starting point until cell containing target point is reached.
“Jump & Walk” Approach

- Use data structure to quickly determine “good” starting point
- Using cell information, walk across cells from starting point until cell containing target point is reached.

query point

first guess
“Jump & Walk” Approach

- Use data structure to quickly determine “good” starting point
- Using cell information, walk across cells from starting point until cell containing target point is reached.
“Jump & Walk” Approach

- Use data structure to quickly determine “good” starting point
- Using cell information, walk across cells from starting point until cell containing target point is reached.
“Jump & Walk” Approach

- Use data structure to quickly determine “good” starting point
- Using cell information, walk across cells from starting point until cell containing target point is reached.
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level.
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level.
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level
Octree

- Octrees (resp. quadtreese) regularly subdivide space along each coordinate axis at each level
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level.
Octree

- Octrees (resp. quadtrees) regularly subdivide space along each coordinate axis at each level
Octree

✓ Widely used in practice
✓ Easy to implement
✓ Allows for local refinement

- Inefficient when cells are skinny
  - large depth value required
  - too many cells per bucket
  - many empty buckets
k-D Tree

- Split spatial domain along successive axis-parallel hyperplanes at data point
  - Equal number of points in each half space
  - balanced tree (optimize overall height)
k-D Tree

• Split spatial domain along successive axis-parallel hyperplanes at data point
  • Equal number of points in each half space
  • balanced tree (optimize overall height)
k-D Tree

- Split spatial domain along successive axis-parallel hyperplanes at data point
  - Equal number of points in each half space
  - Balanced tree (optimize overall height)
k-D Tree

• Split spatial domain along successive axis-parallel hyperplanes at data point
  • Equal number of points in each half space
  • balanced tree (optimize overall height)
k-D Tree

• Split spatial domain along successive axis-parallel hyperplanes at data point
  • Equal number of points in each half space
  • balanced tree (optimize overall height)
k-D Tree

- Split spatial domain along successive axis-parallel hyperplanes at data point
  - Equal number of points in each half space
  - balanced tree (optimize overall height)
k-D Tree

Alternating split

Adaptive split
Outline

- Mesh types
- Interpolation
- Spatial queries
- Common pre-processing tasks
Smoothing

- Implemented as weighted average over a neighborhood
- Smoothing is useful to:
  - reduce noise in data
  - filter out small scale structures (visual clutter)
Smoothing

• If grid is uniform or rectilinear, smooth convolution kernels from image processing are applicable:
  • interpolation
  • approximation

from G. Kindlmann’s PhD thesis
Smoothing

- If grid is uniform or rectilinear, smooth convolution kernels from image processing are applicable.
Smoothing

original 3x3 DOX 7x7 DOX
Smoothing
Smoothing

Unstructured grids

• **Umbrella operator:**
  • Local averaging of surrounding values
  • Applied component-wise for multidimensional data
  
  \[
  f_{k+1}(P_i) = (1 - \omega) f_k(P_i) + \frac{\omega}{|N_1(P_i)|} \sum_{j \in N_1(P_i)} f_k(P_j)
  \]
  
  • Isotropic
  • Iterative
  • Fast!
Smoothing

Unstructured grids

• Take proximity of vertices into account (geometric similarity): closer = more weight

\[ f_{k+1}(P_i) = (1 - \omega)f_k(P_i) + \frac{\omega}{\sum_{j \in N_1(P_i)} \omega_{ij}} \sum_{j \in N_1(P_i)} \omega_{ij} f_k(P_j) \]

\[ \omega_{ij} = \omega(||P_iP_j||) \]

• Non-uniform point distribution, edges of embedded surfaces
Smoothing

Unstructured grids

• Take difference between values into account (data similarity)

\[
f_{k+1}(P_i) = (1 - \omega)f_k(P_i) + \frac{\omega}{\sum_{j \in N_1(P_i)} \omega_{ij}} \sum_{j \in N_1(P_i)} \omega_{ij} f_k(P_j)
\]

\[
\omega_{ij} = \psi(|f_k(P_i) - f_k(P_j)|)
\]

• Preserve strong gradients
Smoothing

Unstructured grids

• Combine both criteria (cf. robust statistics)

\[ f_{k+1}(P_i) = (1 - \omega) f_k(P_i) + \frac{\omega}{\sum_{j \in N_1(P_i)} \omega_{ij}} \sum_{j \in N_1(P_i)} \omega_{ij} f_k(P_j) \]

\[ \omega_{ij} = \phi(||P_i - P_j||) \psi(||f_k(P_i) - f_k(P_j)||) \]

• Preserve strong gradients
Derivative Computation

Cell-wise: derive interpolating function

• Linear interpolation (tri’s, tet’s): constant value

\[
\phi(P) = \sum_{i=1}^{4} \beta_i(P) \phi_i
\]

\[
\nabla \phi(P) = \sum_{i=1}^{4} \nabla \beta_i \phi_i
\]
Derivative Computation

Cell-wise: derive interpolating function

- Bilinear (quads): value depends on local coordinates
- Trilinear (hexahedra, wedges, pyramids): same
Derivative Computation

Point-wise

• Weighted combination of surrounding cell-wise derivatives

• Can be used iteratively for higher-order derivatives (smoothing effect)

• Analytical derivatives of polynomial fit of surrounding values (Taylor expansion)
Resampling

- 2D slice out of 3D volume
- Approximate original data over more convenient structure
- Regular (structured) grid:
  - Lower memory requirements
  - Higher reconstruction quality (derivation, convolution)
  - Suitable for GPU-based algorithms
  - Allows for multi-resolution representation
Resampling

• Challenging with unstructured grids
  • Subsampling (small cells): aliasing
  • Upsampling (big cells): interpolation artifacts

• Subsample after smoothing original data

• Upsample smooth interpolation of the original data
  • convolution kernels on rectilinear grids
  • scattered data interpolation on unstructured grids