Fields, Meshes, and Interpolation (Part 2)

Lecture #3
Hank Childs, University of Oregon
Project #1

• Goal: write a specific image
• Due: “Friday Jan 10th” → “6am Saturday Jan 11th”
• % of grade: 2%
• Goal: get multi-platform issues shaken out ASAP.
• Experience last year was pretty good.

Worth 2% of your grade

Assignment:
1) Download, build, and install VTK.
2) Download and install CMake. Use version 3.X
3) Download the file called data.vtk
4) Make directory called “project1”
5) Download file project1.cxx and CMakeLists.txt from class website and copy them into directory project1
6) Update the VTK_DIR variable in CMakeLists.txt to point to the path of the VTK you just installed.
7) Run CMake. This will create build files.
8) Compile the program. For Unix/Mac, this means “make”
9) Run the program. (How to run is platform dependent ... on Linux and Mac, a binary gets generated and you invoke it.)
10) Submit a screenshot of the working program via Canvas
Outline

• 6 Slide Review
• The Data We Will Study
  — Overview
  — Fields
  — Meshes
  — Interpolation
• Images & color: brief introduction
Outline

• 6 Slide Review
• The Data We Will Study
  — Overview
  — Fields
  — Meshes
  — Interpolation
• Images & color: brief introduction
Scalar Fields

• Defined: associate a scalar with every point in space

• What is a scalar?
  – A: a real number

• Examples:
  – Temperature
  – Density
  – Pressure

The temperature at 41.2324° N, 98.4160° W is 66F.

Fields are defined at every location in a space (example space: USA)
Vector Fields

- Defined: associate a vector with every point in space.
- What is a vector?
  - A: a direction and a magnitude
- Examples:
  - Velocity
    - The velocity at location (5, 6) is (-0.1, -1)
    - The velocity at location (10, 5) is (-0.2, 1.5)

Typically, 2D spaces have 2 components in their vector field, and 3D spaces have 3 components in their vector field.
An example mesh
Anatomy of a computational mesh

- Meshes contain:
  - Cells
  - Vertices
- This mesh contains 3 cells and 13 vertices
- Pseudonyms:
  - Cell == Element == Zone
  - Point == Vertex == Node
Rectilinear meshes

- Rectilinear meshes are easy and compact to specify:
  - Locations of X positions
  - Locations of Y positions
  - 3D: locations of Z positions

- Then: mesh vertices are at the cross product

- Example:
  - X={0,1,2,3}
  - Y={2,3,5,6}
Definition: dimensions

• A 3D rectilinear mesh has:
  – $X = \{1, 3, 5, 7, 9\}$
  – $Y = \{2, 3, 5, 7, 11, 13, 17\}$
  – $Z = \{1, 2, 3, 5, 8, 13, 21, 34, 55\}$

• Then its dimensions are $5 \times 7 \times 9$
Outline

• 6 Slide Review

• The Data We Will Study
  — Overview
  — Fields
  — Meshes
  — Interpolation

• Images & color: brief introduction
• We are beginning to understand meshes ...
• ... but how do we represent them in memory?
How can we represent this mesh?

(let’s make a C++ class and use floats to store data)
What if there is a field on the mesh?

→ assume there is a temperature value at each vertex

(let’s extend our C++ class)
What if there are two fields on the mesh?

→ now assume there is a temperature value and a pressure value at each vertex

(let’s extend our C++ class again)
Decisions

• Enumerate all vertices or just locations in X and Y
• Row major versus column major
• \((T_1, P_1, T_2, P_2, T_3, P_3, ..., T_N, P_N)\) vs. \((T_1, T_2, ..., T_N) \& (P_1, P_2, ..., P_N)\)

• Some choices are better, some are worse, some are neutral
• Whatever choices we make establish our convention
• The following slides show the conventions we will use for this course
Two schemes for indexing points: “logical point indices” and “point indices”

<table>
<thead>
<tr>
<th>Logical point indices</th>
<th>Point indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 1.5 2.5 3.5 4.5 5.5</td>
<td>30 31 32 33 34 35</td>
</tr>
<tr>
<td>0.4 1.4 2.4 3.4 4.4 5.4</td>
<td>24 25 26 27 28 29</td>
</tr>
<tr>
<td>0.3 1.3 2.3 3.3 4.3 5.3</td>
<td>18 19 20 21 22 23</td>
</tr>
<tr>
<td>0.2 1.2 2.2 3.2 4.2 5.2</td>
<td>12 13 14 15 16 17</td>
</tr>
<tr>
<td>0.1 1.1 2.1 3.1 4.1 5.1</td>
<td>6 7 8 9 10 11</td>
</tr>
<tr>
<td>0.0 1.0 2.0 3.0 4.0 5.0</td>
<td>0 1 2 3 4 5</td>
</tr>
</tbody>
</table>

What would these indices be good for?
How to Index Points

• **Our goal**: define a bijective function, $F$, between two sets

  – Set 1 (logical point indices): \{(i,j,k): 0 \leq i < nX, 0 \leq j < nY, 0 \leq k < nZ\}
  
  – Set 2 (point indices): \{0, 1, ..., nPoints-1\}

Bijective: for every element in set 1, there is an element in set 2. And vice-versa.

Note: we will focus on 2D rectilinear meshes for a bit.
How to Index Points

• Many possible conventions for indexing points and cells.

• Most common variants:
  – X-axis varies most quickly
  – X-axis varies most slowly

\[
\begin{array}{cccccccc}
0,5 & 1,5 & 2,5 & 3,5 & 4,5 & 5,5 \\
0,4 & 1,4 & 2,4 & 3,4 & 4,4 & 5,4 \\
0,3 & 1,3 & 2,3 & 3,3 & 4,3 & 5,3 \\
0,2 & 1,2 & 2,2 & 3,2 & 4,2 & 5,2 \\
0,1 & 1,1 & 2,1 & 3,1 & 4,1 & 5,1 \\
0,0 & 1,0 & 2,0 & 3,0 & 4,0 & 5,0 \\
\end{array}
\quad
\begin{array}{cccccccc}
30 & 31 & 32 & 33 & 34 & 35 \\
24 & 25 & 26 & 27 & 28 & 29 \\
18 & 19 & 20 & 21 & 22 & 23 \\
12 & 13 & 14 & 15 & 16 & 17 \\
06 & 07 & 08 & 09 & 10 & 11 \\
00 & 01 & 02 & 03 & 04 & 05 \\
\end{array}
\]
Bijective function for rectilinear meshes for this course

```c
int GetPoint(int i, int j, int nX, int nY)
{
    return j*nX + i;
}
```
Bijective function for rectilinear meshes for this course

```c
int *GetLogicalPointIndex(int point,
                          int nX, int nY)
{
    int rv[2];
    rv[0] = point % nX;
    rv[1] = (point/nX);
    return rv; // terrible code!!
}
```
int *GetLogicalPointIndex(int point, int nX, int nY)
{
    int    rv[2];
    rv[0] = point % nX;
    rv[1] = (point/nX);
    return rv;  // TERRIBLE CODE!
}
Quiz Time #2

- A mesh has dimensions 6x8.
- What is the point index for (3,7)? = 45
- What are the logical indices for point 37? = (1,6)

```c
int GetPoint(int i, int j,
    int nX, int nY)
{
    return j*nX + i;
}

int *GetLogicalPointIndex(int point,
    int nX, int nY)
{
    int rv[2];
    rv[0] = point % nX;
    rv[1] = (point/nX);
    return rv; // terrible code!!
}
```
Quiz Time #3

- A vector field is defined on a mesh with dimensions 100x100
- The vector field is defined with double precision data.
- How many bytes to store the vector field?

\[ = 100 \times 100 \times 2 \times 8 = 160,000 \]
Bijective function for rectilinear meshes for this course

```c
int GetCell(int i, int j, int nX, int nY)
{
    return j*(nX-1) + i;
}
```
Bijective function for rectilinear meshes for this course

```c
int *GetLogicalCellIndex(int cell, int nX, int nY)
{
    int rv[2];
    rv[0] = cell % (nX - 1);
    rv[1] = (cell / (nX - 1));
    return rv; // terrible code!!
}
```
Outline

• 6 Slide Review
• The Data We Will Study
  — Overview
  — Fields
  — Meshes
  — Interpolation
• Images & color: brief introduction
Goal: have data at some points & want to interpolate data to any location
Linear Interpolation for Scalar Field $F$
Linear Interpolation for Scalar Field $F$

- General equation to interpolate:
  - $F(X) = F(A) + t*(F(B)-F(A))$
- $t$ is proportion of $X$ between $A$ and $B$
  - $t = (X-A)/(B-A)$
Quiz Time #4

• F(3) = 5, F(6) = 11
• What is F(4)?  = 5 + (4-3)/(6-3)*(11-5) = 7

• General equation to interpolate:
  – F(X) = F(A) + t*(F(B)-F(A))
• t is proportion of X between A and B
  – t = (X-A)/(B-A)
Bilinear interpolation for Scalar Field $F$

$F(0,0) = 10$
$F(1,0) = 5$
$F(1,1) = 6$
$F(0,1) = 1$

What is value of $F(0.3, 0.4)$?

$F(0.3, 1) = 2.5$
$F(0.3, 0) = 8.5$

$F(0.3, 0.4) = 6.1$

Idea: we know how to interpolate along lines. Let’s keep doing that and work our way to the middle.

- General equation to interpolate:
  
  $$F(X) = F(A) + t \times (F(B) - F(A))$$
Interpolation for triangle meshes

- Two issues:
  - (1) how to locate triangle that contains point
    - (discuss in 5 slides)
  - (2) how to interpolate to value within triangle
    - (discuss now)
Idea #1

• More bilinear interpolation
Idea #1 (cont’d)

- Different triangle, similar idea…
Idea #2: Barycentric Coordinates

\[ V(P) = V(A1) \cdot t1 + V(A2) \cdot t2 + V(A3) \cdot t3 \] / (t1 + t2 + t3)
Cell location

• Problem definition: you have a physical location (P). You want to identify which cell contains P.

It is easy to identify the cell that contains P with our eyes. But more work to instruct a computer to do it. What are your thoughts?
Cell location idea #1 (bad)

• Iterate over every cell
• Check if each cell contains P

• Setup time: zero
• Search time: $O(N)$, where $N$ is the number of cells
• Search time for $M$ queries: $O(M \times N)$
Cell location idea #2 (good)

- Build “quadtree” data structure
  - (see next slide)
- Takes time to build, but then search is cheap

- Setup time: $O(N \log N)$, where $N$ is the number of cells
- Search time: $O(\log N)$
- Search time for $M$ queries:
  - $O(M \cdot \log N) + O(N \log N)$
Cell location idea #2 (good)
Comparing ideas

• Bad idea: $O(M*N)$

• Good idea:
  – $O(N*\log N) + O(M*\log N)$
  – $= O((N+M)*\log N)$

• “Bad idea” is actually better if $M$ very small
Project 2: Field Evaluation

- Assigned today, prompt online
- Due Sunday January 19 midnight (→ 6am January 20\textsuperscript{th})
- Worth 5% of your grade
- I provide:
  - Code skeleton online
  - Correct answers provided
- What you upload to Canvas? ... your source code
- Note: Project 3 coming on Thursday.
Project 2: Field Evaluation

• Basic idea: for point P, find F(P)
• Important: will be on 2D rectilinear meshes, so cell location is easier
• Strategy in a nut shell:
  – Find cell C that contains P
  – Find C’s 4 vertices, V0, V1, V2, and V3
  – Find F(V0), F(V1), F(V2), and F(V3)
  – Find locations of V0, V1, V2, and V3
  – Perform bilinear interpolation to location P
Cell location for project 2

- Traverse X and Y arrays and find the logical cell index.
  - X={0, 0.05, 0.1, 0.15, 0.2, 0.25}
  - Y={0, 0.05, 0.1, 0.15, 0.2, 0.25}

- (Quiz) what cell contains (0.17,0.08)?
  = (3,1)
Facts about cell (3,1)

• It’s cell index is 8.
• It contains points (3,1), (4,1), (3,2), and (4,2).
• Facts about point (3,1):
  – It’s location is (X[3], Y[1])
  – It’s point index is 9.
  – It’s scalar value is F(9).
• Similar facts for other points.
• → we have enough info to do bilinear interpolation
Big O For Cell Location in Project #2

• You should exploit the properties of rectilinear grid.
  – Search along X coordinates for X-position
  – Search along Y coordinates for Y-position
• Setup time: none
• Search time: \( O(\sqrt{N}) + O(\sqrt{N}) = O(\sqrt{N}) \)
What’s in the code skeleton

• Implementations for:
  – GetNumberOfPoints
  – GetNumberOfCells
  – GetPointIndex
  – GetCellIndex
  – GetLogicalPointIndex
  – GetLogicalCellIndex

  { Our bijective function

  – “main”: set up mesh, call functions, create output
What’s not in the code skeleton

```c
// pt: a two-dimensional location
// dims: an array of size two.
//      The first number is the size of the array in argument X,
//      the second the size of Y.
// X: an array (size is specified by dims).
//    This contains the X locations of a rectilinear mesh.
// Y: an array (size is specified by dims).
//    This contains the Y locations of a rectilinear mesh.
// F: a scalar field defined on the mesh. Its size is dims[0]*dims[1].
float EvaluationFunction(const float *pt, const int *dims,
                        const float *X, const float *Y, const float *F)
{
    return 0; // IMPLEMENT ME!!
}
```

... and a few other functions you need to implement
Cell-centered data
Outline

• 6 Slide Review

• The Data We Will Study
  – Overview
  – Fields
  – Meshes
  – Interpolation

• Images & color: brief introduction
Color
“Today’s Lecturer”: Kristi Potter

Kristi Potter

• Previous:
  – B.S., Univ. of OR: CIS & Fine Arts
  – Ph.D., Univ. of Utah
  – Professional Researcher, Univ. of Utah

• Current:
  – Scientific Programmer, CASSPR, Univ. of OR
  – Courtesy position, CIS
  – Co-founder, CDUX
  – Campus “visualization evangelist”

• Research expertise:
  – Scientific visualization, esp. visualization of ensembles and uncertainty visualization
Physics of Light
The Electromagnetic Spectrum

- **Gamma Rays**: $1 \times 10^{-14}$
- **X-Rays**: $1 \times 10^{-12}$
- **Ultraviolet Rays**: $1 \times 10^{-8}$
- **Infrared Rays**: $1 \times 10^{-4}$
- **Radar**: $1 \times 10^{-2}$
- **FM**: $1 \times 10^2$
- **TV**: $1 \times 10^4$
- **Shortwave**: $1 \times 10^4$
- **AM**:

Wavelength (in meters)

- **Visible Light**
- **Wavelength (in meters)**: 4 x $10^{-7}$ to 7 x $10^{-7}$

High Energy → Low Energy
The Visible Spectrum

human visual system sensitive to ~380-780 nm
Isaac Newton

- white light is a combination of all colors of the spectrum
- black is the absence of visible spectrum wavelengths
- separation of visible light into colors is called \textit{dispersion}

- objects appear colored by the character of the light they reflect
Images
Background on Images

• Definitions:
  – Image: 2D array of pixels
  – Pixel: A minute area of illumination on a display screen, one of many from which an image is composed.

• Pixels are made up of three colors: Red, Green, Blue (RGB)

• Amount of each color scored from 0 to 1
  – 100% Red + 100% Green + 0% Blue = Yellow
  – 100% Red + 0% Green + 100 %Blue = Purple
  – 0% Red + 100% Green + 100% Blue = Cyan
  – 100% Red + 100% Blue + 100% Green = White
Background on Images

• Colors are 0->1, but how much resolution is needed? How many bits should you use to represent the color?
  – Can your eye tell the difference between 8 bits and 32 bits?
  – ➔ No. Human eye can distinguish ~10M colors.
  – 8bits * 3 colors = 24 bits = ~16M colors.

• Red = (255,0,0)
• Green = (0,255,0)
• Blue = (0,0,255)
How to organize a struct for an Image (i.e., 3D arrays)

• 3D array: width * height * 3 color channels

• Color:
  – Choice 1: RGB struct
    • struct rgb { unsigned char r, g, b; };
    • int npixels = width*height;
    • struct RGB *buffer = new RGB[npixels];
    • int p = 21456; // random pixel in the buffer
      buffer[0].r = 255; buffer[0].g = 0; buffer[0].b = 0;
  – Choice 2: just 3 unsigned chars
How to organize a struct for an Image (i.e., 3D arrays)

• 3D array: width * height * 3 color channels

• Color:
  – Choice 1: RGB struct
  – Choice 2: just 3 unsigned chars
    • int npixels = width*height;
    • unsigned char *buffer = new unsigned char [3*npixels];
    • int p = 21456; // random pixel in the buffer
    • buffer[3*p+0] = 255; buffer[3*p+1] = 0;
    • buffer[3*p+2] = 0;
For project 3, I am doing the management of the buffer
  — I do choice 2
But you will write functions that work on one pixel
void AssignValue (unsigned char *pixel) {
    pixel[0] = 255; pixel[1] = 0; pixel[2] = 0; }
My code:
  — for (int i = 0 ; i < npixels ; i++) AssignValue(buffer[3*i]);
Your Amazing Eyes
Crazy numbers about your eyes (with possibly some exaggeration)

• What is the pixel resolution of your eyes?
  50*MegaPixels
  (* = different parts of the eye work differently; 50M pixel is an aggregation)

• What is the frequency your eyes take in information?
  ~20* Hertz
  (* = for VR, 90Hz is sometimes needed. For video games, 30Hz almost always sufficient)

50MegaPixels x 20HZ → your eyes can take in 1GB of data per second
This is why visualization is king for understanding data.
Pseudocolor plot

• First visualization technique we will learn
• Idea: take a scalar field on a mesh and transform it to colors
• Two elements:
  – Define a map
  – Apply the map
Pseudocolor plot

• First visualization technique we will learn
• Idea: take a scalar field on a mesh and transform it to colors
• Two elements:
  – Define a map
  – Apply the map
Color map example ("discrete")

- Example below defines the color for scalar values between 0 and 1.29
- Grouped in 13 ranges ("discrete")
Color map example ("continuous")

- Example below defines the color for scalar values between -0.10 and 0.10
- No groupings ("continuous")
- This color map appears to have a fixed number of colors, and then blends between those colors
  - Red, yellow, green, cyan, blue
  - (This is not immediately obvious)
Pseudocolor plot

• First visualization technique we will learn
• Idea: take a scalar field on a mesh and transform it to colors
• Two elements:
  – Define a map
  – Apply the map
Pseudocolor plot

• Defined:
  – Create a mapping, $M$, from scalar values to colors
  – For each pixel:
    • Evaluate $V$, the scalar field at that pixel location
    • Obtain color $C$, by applying $M$ to $V$
      – $C = M(V)$
    • Assign the pixel color to be $C$
Example pseudocolor plot (discrete color map)

Mean Daily Average Temperature
Source: Climate Atlas of the United States
Pseudocolor in practice

• The normal way:
  – Data set is composed of triangles
  – Apply color map to each vertex in the triangle
  – Tell graphics hardware to render each triangle (including color at each of three vertices)
  – Graphics hardware does the work of interpolating color at each pixel