Upcoming Schedule

• 12/6: unstructured grids
• Faculty fireside (parallel visualization):
  – Week of 12/9
• Final: Thurs 12/12, 3:15PM, Rm 220
Announcements

• Final projects:
  – Custom projects (~20):
    • bring up to 5 min presentation
      – Auxiliary grading time possible
  – Movies (~10):
    • prepare to show off your movie
      – ~3 mins
  – total ~2 hours (might be a little long)
• Private (offline) demonstrations if necessary
Announcements

• Kristi, Erik, and I will judge them, award best project & best movie
• Location: Rm 220 (colloquium room)
• Attendance is mandatory (& graded), even for ray casters
• Late policy:
  – 80% of the points if you are two days late,
  – 60% of the points if you are four days late,
  – 50% of the points any time thereafter
• Final project worth 35% of your grade
An example mesh

Why do you think the triangles change size?
Types of Meshes

- Curvilinear
- Adaptive Mesh Refinement
- Unstructured
Curvilinear Mesh

- Logically rectilinear
  - Each cell has an \((i, j)\)
  - Always left, right, bottom, top neighbor (unless on boundary)

- Points can be anywhere ... as long as the cells don’t overlap

VTK calls this “vtkStructuredGrid”
Curvilinear Mesh

- A curvilinear mesh has 5x5 cells and a cell-centered variable stored.
- Quiz: how many bytes to store this data set if all data is in single precision floating point?

\[ 2 \text{ ints} + 6 \times 6 \text{ floats (points)} + 5 \times 5 \text{ floats (variable)} = 2 \times 4 + 36 \times 4 + 25 \times 4 = 63 \times 4 = 272 \text{ bytes} \]
Example unstructured mesh

- Meshes contain:
  - Cells
  - Points
- This mesh contains 3 cells and 13 vertices
- Pseudonyms:
  - Cell == Element == Zone
  - Point == Vertex == Node
If we stored each cell like this, how many bytes would it take? (assume single precision)

A: 1 int (# cells), 3 ints (# pts per cell), 24+24+15 floats = 268 bytes

Let’s call this the “explicit” scheme

• Cell 0:
  \[
  \{ (0, 0, 0), // 10
    (1, 0, 0), // 11
    (1, 1, 0), // 8
    (0, 1, 0), // 7
    (0, 0, 1), // 4
    (1, 0, 1), // 5
    (1, 1, 1), // 2
    (0, 1, 1) } // 1
  \]
Example unstructured mesh

- **Pts:**
  - \{(0, 1, 1), (1, 1, 1), (2, 1, 1), (0, 0, 1), (1, 0, 1), (2, 0, 1), (0, 1, 0), (1, 1, 0), (2, 0, 0), (0, 0, 0), (1, 0, 0), (2, 0, 0), (2.5, 0.5, 0.5)\}

- **Cells:**
  - 3 cells
    - 1\textsuperscript{st} cell: hexahedron – (10, 11, 8, 7, 4, 5, 2, 1)
    - 2\textsuperscript{nd} cell: hexahedron – (11, 12, 9, 8, 5, 6, 3, 2)
    - 3\textsuperscript{rd} cell: prism – (13, 3, 6, 12, 9)

If we stored each cell like this, how many bytes would it take? (assume single precision)

A: 1 int (# pts), 1 int (# cells), 3 ints (# cell type), 13*3 floats (pts), 8+8+5 ints = 260 bytes
Comparing unstructured mesh storage schemes

• Hexahedral meshes: each internal point incident to 8 cells
  – Explicit scheme:
    • represent that point 8 times: 24 floats for storage
  – Connectivity scheme:
    • represent that point once in point list, 8 times in connectivity list: 3 floats + 8 ints

Further benefit to connectivity scheme is in finding exterior faces.
Finding external faces: motivation

• Interval volume, clip:
  – Take data set (rectilinear, unstructured, or other) and produce unstructured mesh
  – When rendering, only want to render the faces on the outside (the inside aren’t visible)

Question: what proportion of faces are exterior?

Question: how to find exterior faces?
Finding external faces: algorithm

• For each face, count how many cells are incident.
  – If “1”, then external
  – If “2”, then interior

Question: why does this work?
Finding exterior faces: algorithm

• Estimate # of faces (ncells * 6 / 2)
• Double that number
• Create a hash table of that size
• For each cell C
  – For each face F of C
    • Create hash index for F based on connectivity indices
    • Search hash table
      – If F already there, remove F from hash
      – If face not there, add F to hash

• All faces in hash are exterior
Interpolation for arbitrary cells: tetrahedrons

• Assume tetrahedron $T$, point $P$ in $T$
• Goal: calculate $F(P)$
Interpolation for arbitrary cells: tetrahedrons

- Assume tetrahedron $T$, point $P$ in $T$
- Goal: calculate $F(P)$

Set up parametric coordinate system
Interpolation for arbitrary cells: tetrahedrons

- Assume tetrahedron T, point P in T
- Goal: calculate F(P)

\[ P = P_0 + aR + bS + cT \]

This is a 3x3 matrix solve. This matrix is invertible since R, S, T form a basis.

Calculate parametric coordinates \((a, b, c)\)
Interpolation for arbitrary cells: tetrahedrons

• Assume tetrahedron T, point P in T
• Goal: calculate F(P)

\[
P = P_0 + aR + bS + cT
\]

\[
F(P) = \text{sum}(W_i * F(P_i))
\]

\[
W_0 = 1 - a - b - c
\]

\[
W_1 = a
\]

\[
W_2 = b
\]

\[
W_3 = c
\]
General idea

• Set up parametric coordinate system
• Calculate parametric coordinates for P
• Calculate $F(P)$ as $\sum(W_i \cdot F(P_i))$
  – Weights $W_i$ can get pretty tricky.

VTK book has weights & good description in Ch. 8.2.
How to do contouring

• Basically the same:
  – Iterate over cells
  – Identify case
  – Lookup case in table
  – Create resulting geometry

• Difference:
  – New tables for each cell type
How to do ray casting

• Basically the same:
  – Cast rays for every pixel on the screen
  – Sample along rays
  – Apply transfer function
  – Composite front to back
  – Assign color to image

• Differences:
  – Sampling gets hard!
    • Which cell contains a sample point?
      – Need smart data structures ....
      – .... Or a way to transform data to make it easy
How to do particle advection

• Basically the same:
  – Start with a seed location
  – Evaluate velocity
  – Displace particle
  – Repeat until termination criteria reached

• Differences:
  – Evaluating velocity different x2:
    • Now a harder proposition: which cell contains particle?
    • Now more math: how to LERP velocity?
Adaptive Mesh Refinement

• Put resolution where you want it...
• ... but simpler than unstructured (indexing & interpolation) and cheaper (memory)
• Problems:
  – Everything great, except at the boundaries....
Looking Back: This Class

• I hope you learned: the theory of scientific visualization
  – Visualization is for 3 main use cases:
    • Communication
    • Confirmation
    • Exploration
      – Enables the scientific method
  – Many visual metaphors for representing data
    • What are the tools?
    • How to choose the right tool from the toolbox?
    • What are the systems that support the tools?
Looking Back: This Class

• I hope you learned: how to be a better programmer
  – These projects were challenging
    • I hope you enjoyed them
    • I hope you learned from them

Brag about these … put thumbnails on your resumes
My Unsolicited Advice

• Self-actualize: to realize one’s full potential
  – If you are not excited about your job, find a new one (*)
    • Repeat until you are happy or until you are convinced this is the wrong line of work for you.
    • (*) = Life will get in the way sooner than you think

• You have all done extremely well in this class ... seek that job with significant technical responsibility.

• Learn about yourself
Thank you!!

- It has been a privilege to teach this class and get to know you all.
  - I appreciate the earnest effort you have put forth
  - I appreciate you bearing with me through many mistakes
    - (The good news is that we will now have qualified TAs to choose from the next time we offer this course!!)