four control points, two of which are knots
more intuitive definition than derivatives
curve will always remain within convex hull (bounding region) defined by control points
Review: Comparing Hermite and Bézier

Hermite

Bézier

Review: Sub-Dividing Bézier Curves

- find the midpoint of the line joining \( M_{012} \) and \( M_{123} \)
  - call it \( M_{0123} \)

Review: de Casteljau’s Algorithm

- can find the point on Bézier curve for any parameter value \( t \) with similar algorithm
  - for \( t=0.25 \) instead of taking midpoints take points 0.25 of the way

Review: Continuity

- piecewise Bézier: no continuity guarantees

- continuity definitions
  - \( C^0 \): share join point
  - \( C^1 \): share continuous derivatives
  - \( C^2 \): share continuous second derivatives

Review: B-Spline

- \( C^0 \), \( C^1 \), and \( C^2 \) continuous
- piecewise: locality of control point influence

Virtual Trackball
Virtual Trackball
- interface for spinning objects around
- drag mouse to control rotation of view volume
- rolling glass trackball
- center at screen origin, surrounds world
- hemisphere “sticks up” in z, out of screen
- rotate ball = spin world

Virtual Trackball
- know screen click: (x, 0, z)
- want to infer point on trackball: (x, y, z)
- ball is unit sphere, so \( |x, y, z| = 1.0 \)
- solve for \( y \)

Trackball Rotation
- correspondence:
  - moving point on plane from \((x, 0, z)\) to \((a, 0, c)\)
  - moving point on ball from \(p_1 = (x, y, z)\) to \(p_2 = (a, b, c)\)
- correspondence:
  - translating mouse from \(p_1\) (mouse down) to \(p_2\) (mouse up)
  - rotating about the axis \(n = p_1 \times p_2\)

Trackball Computation
- user defines two points
  - place where first clicked \(p_1 = (x, y, z)\)
  - place where released \(p_2 = (a, b, c)\)
- create plane from vectors between points, origin
- axis of rotation is plane normal: cross product
  \((p_1 - o) \times (p_2 - o): p_1 \times p_2\) if origin = \((0,0,0)\)
- amount of rotation depends on angle between lines
  \(p_1 \cdot p_2 = |p_1||p_2| \cos \theta\)
  \(|p_1 \times p_2| = |p_1||p_2| \sin \theta\)
- compute rotation matrix, use to rotate world

Scientific Visualization

Reading
- FCG Chapter 23
Surface Graphics
- objects explicitly defined by surface or boundary representation
  - mesh of polygons

Volume Graphics
- for some data, difficult to create polygonal mesh
  - voxels: discrete representation of 3D object
  - volume rendering: create 2D image from 3D object
  - translate raw densities into colors and transparencies
  - different aspects of the dataset can be emphasized via changes in transfer functions

Surface Graphics
- pros
  - fast rendering algorithms available
  - hardware acceleration cheap
  - OpenGL API for programming
  - use texture mapping for added realism
- cons
  - discards interior of object, maintaining only the shell
  - operations such cutting, slicing & dissection not possible
  - no artificial viewing modes such as semi-transparencies, X-ray
  - surface-less phenomena such as clouds, fog & gas are hard to model and represent

Volume Graphics
- cons
  - rendering algorithm has high complexity!
  - special purpose hardware costly (~$3K-$10K)

Isosurfaces
- 2D scalar fields: isolines
  - contour plots, level sets
  - topographic maps
- 3D scalar fields: isosurfaces
Volume Graphics: Examples
- anatomical atlas from visible human (CT & MRI) datasets
- industrial CT - structural failure, security applications
- flow around airplane wing
- shockwave visualization: simulation with Navier-Stokes PDEs

Isosurface Extraction
- array of discrete point samples at grid points
- 3D array: voxels
- find contours
  - closed, continuous
  - determined by iso-value
- several methods
  - marching cubes is most common

MC 1: Create a Cube
- consider a cube defined by eight data values

\[ (i,j,k), (i+1,j,k), (i,j+1,k), (i+1,j+1,k), (i,j,k+1), (i+1,j,k+1), (i+1,j+1,k), (i+1,j+1,k+1) \]

MC 2: Classify Each Voxel
- classify each voxel according to whether lies outside the surface (value > iso-surface value)
- inside the surface (value <= iso-surface value)

MC 3: Build An Index
- binary labeling of each voxel to create index

\[ \text{Index: } 00110000, 11101001 \]

MC 4: Lookup Edge List
- use index to access array storing list of edges
- all 256 cases can be derived from 15 base cases
MC 4: Example
- index = 00000001
- triangle 1 = a, b, c

MC 5: Interpolate Triangle Vertex
- for each triangle edge
  - find vertex location along edge using linear interpolation of voxel values

MC 6: Compute Normals
- calculate the normal at each cube vertex
- use linear interpolation to compute the polygon vertex normal
  \[ \begin{align*}
  G_x &= v_{i+1,j,k} - v_{i-1,j,k} \\
  G_y &= v_{i,j+1,k} - v_{i,j-1,k} \\
  G_z &= v_{i,j,k+1} - v_{i,j,k-1}
  \end{align*} \]

MC 7: Render!
- Direct Volume Rendering
  - do not compute surface
- Rendering Pipeline
  - Classify
Classification
- data set has application-specific values
  - temperature, velocity, proton density, etc.
- assign these to color/opacity values to make sense of data
- achieved through transfer functions

Transfer Functions
- map data value to color and opacity

Setting Transfer Functions
- can be difficult, unintuitive, and slow

Rendering Pipeline
- Classify
- Shade

Light Effects
- usually only consider reflected part
Rendering Pipeline

Interpolation
- 2D: given: nearest neighbor, linear
- 1D: given: linear

Volume Rendering Algorithms
- ray casting
  - image order, forward viewing
- splatting
  - object order, backward viewing
- texture mapping
  - object order
  - back-to-front compositing

Ray Traversal Schemes

Ray Traversal - First
- first: extracts iso-surfaces (again!)
Ray Traversal - Average
- average: looks like X-ray

Ray Traversal - MIP
- max: Maximum Intensity Projection
  - used for Magnetic Resonance Angiogram

Ray Traversal - Accumulate
- accumulate: make transparent layers visible

Splatting
- each voxel represented as fuzzy ball
  - 3D gaussian function
  - RGBA value depends on transfer function
- fuzzy balls projected on screen, leaving footprint called splat
  - composite front to back, in object order

Texture Mapping
- 2D: axis aligned 2D textures
  - back to front compositing
  - commodity hardware support
  - must calculate texture coordinates, warp to image plane
- 3D: image aligned 3D texture
  - simple to generate texture coordinates