Review

- Iso-contours in grayscale images and volumes
  - Piece-wise linear representations
    - Polylines (2D) and meshes (3D)
  - Primal and dual methods
    - Marching Squares (2D) and Cubes (3D)
    - Dual Contouring (2D, 3D)
  - Acceleration using trees
    - Quadtree (2D), Octree (3D)
    - Interval trees
Geometry Processing

• Fairing (smoothing)
  – Relocating vertices to achieve a smoother appearance

• Simplification
  – Reducing vertex count

• Deformation
  – Relocating vertices guided by user interaction or to fit onto a target
Fairing a 1D Signal

- Taking out “high frequency” content

(slides materials courtesy of Andy van Dam)
Waveform Analysis

- A signal is the sum of phase-shifted sine curves at various scales (frequencies)

(slides materials courtesy of Andy van Dam)
Frequency Domain

- Plotting the amplitude of sine curves at each frequency

Real domain

Frequency domain

(slides materials courtesy of Andy van Dam)
Signal Filtering

- Low-pass filtering: A “box” filter in the frequency domain
  - Equivalent to convolution in the real domain

(slides materials courtesy of Andy van Dam)
Convolution

- Given two functions, \( f \) (signal) and \( w \) (filter)
  - \( w \) is symmetric: \( w[x] = w[-x] \)
- \((f * w)[a] \) is:
Convolution

- Given two functions, \( f \) (signal) and \( w \) (filter)
  - \( w \) is symmetric: \( w[x] = w[-x] \)

- \((f * w)[a]\) is:
  - Shift \( w \) so that it’s centered at \( a \)
  - Multiply \( f \) with (shifted) \( w \)
  - Find the area under the multiplied curve

\[
(f * w)[a] = \int_{-\infty}^{\infty} f[x] \cdot w[x - a] \, dx
\]
Convolution

- Low-pass (Box) filtering: convolution with $\text{Sinc}[x]$
  - Removes all frequencies greater than $B/2$

\[
w[x] = B \text{Sinc}[Bx]
\]

\[
\text{Sinc}[x] = \frac{\sin(\pi x)}{\pi x}
\]

$B = 1$
Convolution

- Gaussian filtering: convolution with Gaussian function
  - Kernel size $\delta$ (standard deviation) controls the amount of smoothing
  - Repeated filtering using a small kernel is equivalent to a single filtering with a large kernel

$$w[x] = \frac{1}{\sqrt{2\pi}\delta} e^{-\frac{x^2}{2\delta^2}}$$

$\delta = 1$
Discrete Convolution

• $f$, $w$ as two arrays
  - $\{f_{-\infty}, \ldots, f_{\infty}\}$: data samples
  - $\{w_{-\infty}, \ldots, w_{\infty}\}$: weights
• Convolution yields another array

$$(f * w)_i = \sum_{j=-\infty}^{\infty} f_j \times w_{j-i}$$
Discrete Convolution

- $f, w$ as two arrays
  - $\{f_{-\infty}, \ldots, f_{\infty}\}$: data samples
  - $\{w_{-\infty}, \ldots, w_{\infty}\}$: weights
- Convolution yields another array

$$(f * w)_i = \sum_{j=-\infty}^{\infty} f_j \times w_{j-i}$$
Discrete Convolution

- \( f, w \) as two arrays
  - \( \{f_{-\infty}, \ldots, f_{\infty}\} \): data samples
  - \( \{w_{-\infty}, \ldots, w_{\infty}\} \): weights
- Convolution yields another array

\[ (f \ast w)_i = \sum_{j=-\infty}^{\infty} f_j \times w_{j-i} \]
Discrete Convolution

- Example: convolution with a simple Gaussian kernel
  - Weights \( \{w_{-1}, w_0, w_1\} \) are \( \{1/4, 1/2, 1/4\} \)

\[
(f * w)_i = \frac{1}{4} f_{i-1} + \frac{1}{4} f_{i+1} + \frac{1}{2} f_i
\]
Discrete Convolution

- Example: convolution with a simple Gaussian kernel
  - Weights \( \{w_{-1}, w_0, w_1\} \) are \( \{1/4, 1/2, 1/4\} \)
  - Intuitively: the signal at a point is moved to half-way between the signal and the average signals of the two neighboring points.

\[
(f * w)_i = (f_i + f'_i)/2
\]

\[
f'_i = (f_{i-1} + f_{i+1})/2
\]
Curve Fairing

- Reducing “bumpiness” by changing the vertex locations
Curve Fairing

- Curve as a signal
  - The “signal” at each vertex is its location

\( \{x, y\} \)
Curve Fairing

- Fairing by **mid-point averaging**
  - Moving each vertex towards the mid-point of its two neighbors (like discrete Gaussian filtering)
    - \( \lambda \): some value between 0 and 1

\[
p' = (1 - \lambda)p + \lambda\left(\frac{p_1 + p_2}{2}\right)
\]

- Iterative fairing
  - At each iteration, update **all** vertices using locations in the previous iteration
  - A \( \lambda \) close to 1 will create oscillation
    - Typically \( \lambda = 0.5 \)
Curve Fairing

- Drawback
  - The initial shape is shrunk!

100 iterations
200 iterations
400 iterations
Curve Fairing

• **Non-shrinking mid-point averaging** [Taubin 1995]
  
  – Alternate between two kinds of iterations with different $\lambda$
    
    • **Odd** iterations: $\lambda_{\text{odd}} = 0.5$ (positive)
      
      – Shrinking the shape
    
    • **Even** iterations: $\lambda_{\text{even}} = 1/(\kappa - \frac{1}{\lambda_{\text{odd}}})$ (negative)
      
      – $\kappa$ : typically 0.1
      
      – Expanding the shape
Curve Fairing

- The initial shape is no longer shrunk
  - The result converges with increasing iterations
Surface Fairing

- Fairing by **centroid averaging**
  - Moving each vertex towards the centroid of its edge-adjacent neighbors (called the 1-ring neighbors)

\[ p' = (1 - \lambda)p + \lambda \sum_{i=1}^{m} \frac{p_i}{m} \]

- Iterative, **non-shrinking** fairing
  - Alternate between shrinking and expanding
    - Same choices of \( \lambda \) as in 2D
  - Each iteration updates all vertices using locations in the previous iteration
Surface Fairing

- Example: fairing iso-surface of a binary volume
• Implementation Tips
  – At each iteration, keep two copies of locations of all vertices
    • Store the smoothed location of each vertex in another list separate from the current locations
  – Building an adjacency table storing the neighbors of each vertex would be helpful, but not necessary
    • Initialize a temporary centroid as \{0,0,0\} at each vertex, and its neighbor count as 0.
    • For each triangle, add the coordinates of each vertex to the temporary centroids stored at the other two vertices and increment their neighbor count.
      – The neighbor count is twice the actual # of edge neighbors
    • For each vertex, divide the centroid by its neighbor count.