GL8: Computer-Aided Design (CAD)

- History of CAD developments
- Representation of graphical information in a computer
- Modelling systems
History of CAD developments

• From early 1960s: Computer-aided 2D drafting

• Numerical control (NC) machine tools stimulated development of ‘2½D’ CAD systems

• 3D ‘wireframe’ modellers: a natural extension of 2D drafting systems

• 3D surface modellers: define complex ‘skin’ shapes in aero- and auto-industries

• 3D solid modellers: complete representation of solid properties
Computer-aided drafting

- 2D drawings, as on a drawing board
- geometry recorded as points and lines
- not faster to draw, but easier to modify and re-use
2½D CAD systems

- Numerical control (NC) machine tools stimulated the development of ‘2½D’ CAD systems

2D model with depth info; dubbed ‘2 ½ D’

Machined object
3D wireframe modellers

- A natural extension of 2D drafting systems
- Geometry recorded as vertices, edges and limiting elements

Wireframe model

Object to be represented
Interpretation of wireframe model assisted by hidden line removal or suppression.
Hidden line suppression helps remove ambiguity

The isometric projection could represent:

this ... ... or this
A wireframe model cannot represent a solid uniquely.

This wireframe model could equally represent...

this solid ...

or this one
The software must check for validity of solid

- Each vertex has a unique location
- Each vertex associated with $\geq 3$ edges
- Each edge has only 2 vertices
- Each face must have $\geq 3$ edges which form a closed loop

Yes!

A valid face, but not part of a solid

No!

only two!
- not yet a solid
only two edges!
- still not a solid
3 edges at vertex
Computer representation and manipulation of geometry

- Things that are very easy to do with pencil, paper and an eraser can be quite difficult to organise in a computer!
- Simple example. We wish to be able to:
  - add to a list of points on a plane
  - display these points
  - click on a point to delete it
Points on a plane

Display

List in memory

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.32</td>
<td>67.79</td>
</tr>
<tr>
<td>67.06</td>
<td>43.48</td>
</tr>
<tr>
<td>13.02</td>
<td>55.94</td>
</tr>
<tr>
<td>87.53</td>
<td>15.92</td>
</tr>
<tr>
<td>83.77</td>
<td>83.40</td>
</tr>
<tr>
<td>56.14</td>
<td>32.50</td>
</tr>
<tr>
<td>15.81</td>
<td>13.16</td>
</tr>
</tbody>
</table>

Adding to the list is straightforward - until space allocated to list is full!
Picking points for deletion

- Find distance from pick point \((x_p, y_p)\) to each data point \((x_i, y_i)\):
  
  \[ s_i = \sqrt{(x_i - x_p)^2 + (y_i - y_p)^2} \]

  for \(i = 1\) to end_of_list

- Find minimum distance
Some problems ...

Find $\min \left\{ s_i = \sqrt{(x_i - x_p)^2 + (y_i - y_p)^2} \right\}$

• Square-root function is slow to compute for a long list
  – compute just squared distance instead

• Economise further. For each list entry...
  - calculate both $|x_i - x_p|$ and $|y_i - y_p|$  
  - then, square the larger of these
  - if this is larger than the smallest $s^2$ so far, move on...

• For a long list, this is still ‘expensive’
Use a ‘map reference’

- Store ‘pointers’ to x-y data for grid cells
- Identify grid cell of ‘click’
- Compute shortest distance for points in that cell only
A ‘tree’ structure uses memory more efficiently

To store data, divide area:
- if sub-area contains one point, record pointer to x-y data
- else divide again

Search thru tree for point in same cell as click
Data structures

• The previous simple example of management of data for a collection of points illustrates the importance of data structures
  – Speed, accuracy and data capacity are only obtained with some effort
  – Sophisticated data structures and search strategies are required
Representation of straight lines

Familiar *explicit* equation to straight line:

\[ y = mx + c \]

- Problems:
  - for vertical lines, \( m \to \infty \)
  - using alternative \( x = k \) is inconvenient
**Representation of straight lines**

- **Implicit** equation for straight line:
  \[ a_1 x + b_1 y + c_1 = 0 \]
  - handles all slopes

- Normalise by dividing by \( \sqrt{a_1^2 + b_1^2} \):
  \[ ax + by + c = 0 \]

- Then \( a^2 + b^2 = 1 \)
  and \( c \) = perpendicular distance from line to origin
Normalised implicit equation of straight line

Divides plane into two regions

Useful to find on which side of line is a given point 

\((x_p, y_p)\)

Calculate 

\[ p = ax_p + by_p + c \]

This is perp. distance from point to line
Parametric equation for a line

\[ x = x_0 + l \cdot u \]
\[ y = y_0 + m \cdot u \]
\[ z = z_0 + n \cdot u \]

\( u \) is a parameter (the distance along the line from \( P_0 \))

Global coordinate system
Follow up

• Read Bertoline:
  – Chapter 7

• Do problems from Bertoline:
  – Prob 7.1