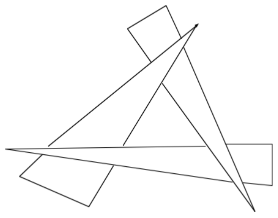
Visibility - painters algorithm

- Algorithm
  - Choose an order for the polygons based on some choice (e.g. depth to a point on the polygon)
  - Render the polygons in that order, deepest one first
- This renders nearer polygons over further.
- Works for some important geometries (2.5D - e.g. VLSI, mazes--but more efficient algorithms exist)
- Doesn't work in this form for most geometries (see figure)



The Z - buffer

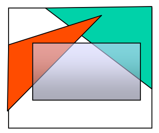
- For each pixel on screen, have a second memory location - called the z-buffer
- Initialize this buffer to a value corresponding to the furthest point possible.
- As a polygon is filled in, compute the depth value of each pixel
  - if depth < z buffer depth, fill in pixel and new depth
  - else disregard
- Typical implementation: Compute Z while scan-converting. A ∂Z for every ∂X is easy to work out.

The Z - buffer

- Advantages:
  - simple; hardware implementation common
  - efficient z computations are easy.
  - ok with lots of surfaces (if there are lots, they tend to be small, and not much difference to this algorithm)
- Disadvantages:
  - over renders - can be slow for very large collections of polygons - may end up scan converting many hidden objects
  - quantization errors can be annoying (not enough bits in the buffer)
  - doesn't help with transparency, or filtering for anti-aliasing.

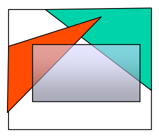
The A - buffer

- For transparent surfaces and filter anti-aliasing:
- Algorithm: filling buffer
  - at each pixel, maintain a pointer to a list of polygons sorted by depth.
  - when filling a pixel:
    - if polygon is opaque and covers pixel, insert into list, removing all polygons farther away
    - if polygon is opaque and only partially covers pixel, insert into list, but don't remove farther polygons


pixel

The A - buffer

- Algorithm: rendering pixels
  - at each pixel, traverse buffer using brightness values in polygons to fill.
  - values are used for either for calculations involving transparency or for filtering for aliasing

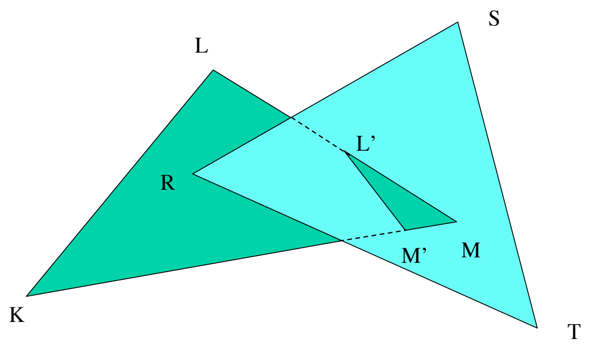

pixel

Scan line algorithm

- Assume (for a moment) that polygons do not intersect one another.
- Observation: on any given scan line, the visible polygon can change only at an edge.
- Algorithm:
  - fill all polygons simultaneously at each scan line, have all edges that cross scan line in AEL
  - keep record of current depth at current pixel
  - use current depth to decide which polygon to use for a span when a new edge is encountered

Scan line algorithm

- To deal with penetrating polygons, split them up



Scan line algorithm

- Advantages:
  - potentially fewer quantization errors (typically more bits available for depth, but this depends)
  - filter anti-aliasing can be made to work.
- Disadvantages:
  - invisible polygons clog AEL and ET. (Can get expensive for complex scenes).