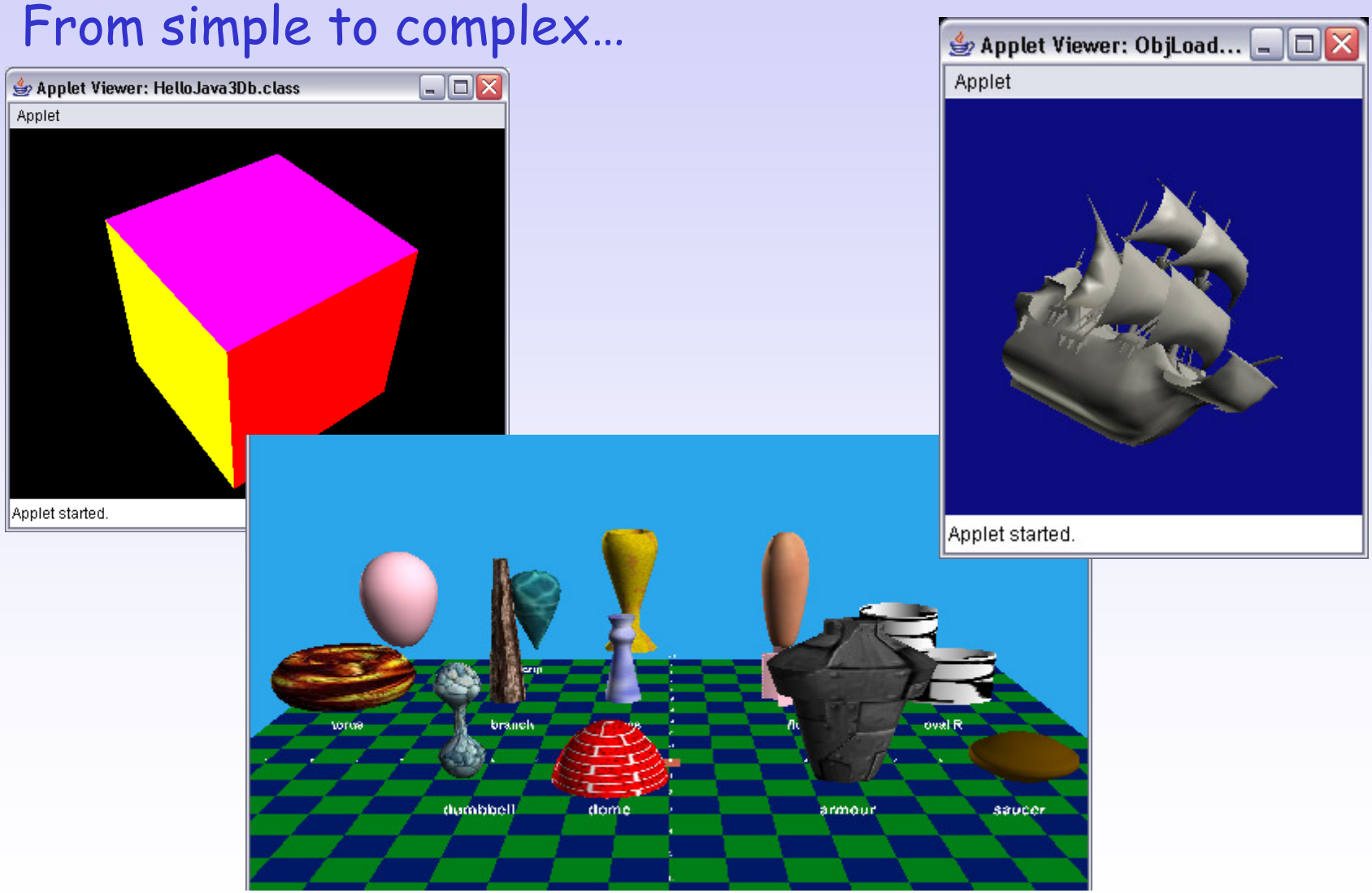
Computer Game Technologies

3D Object Modelling

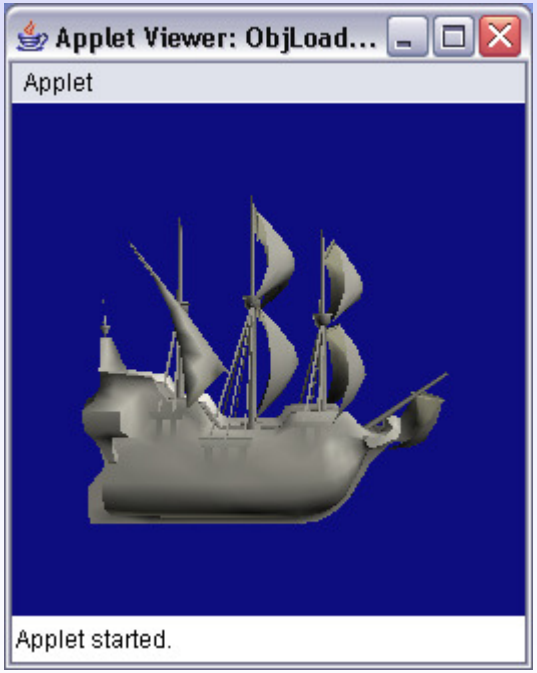
Example 3D Objects

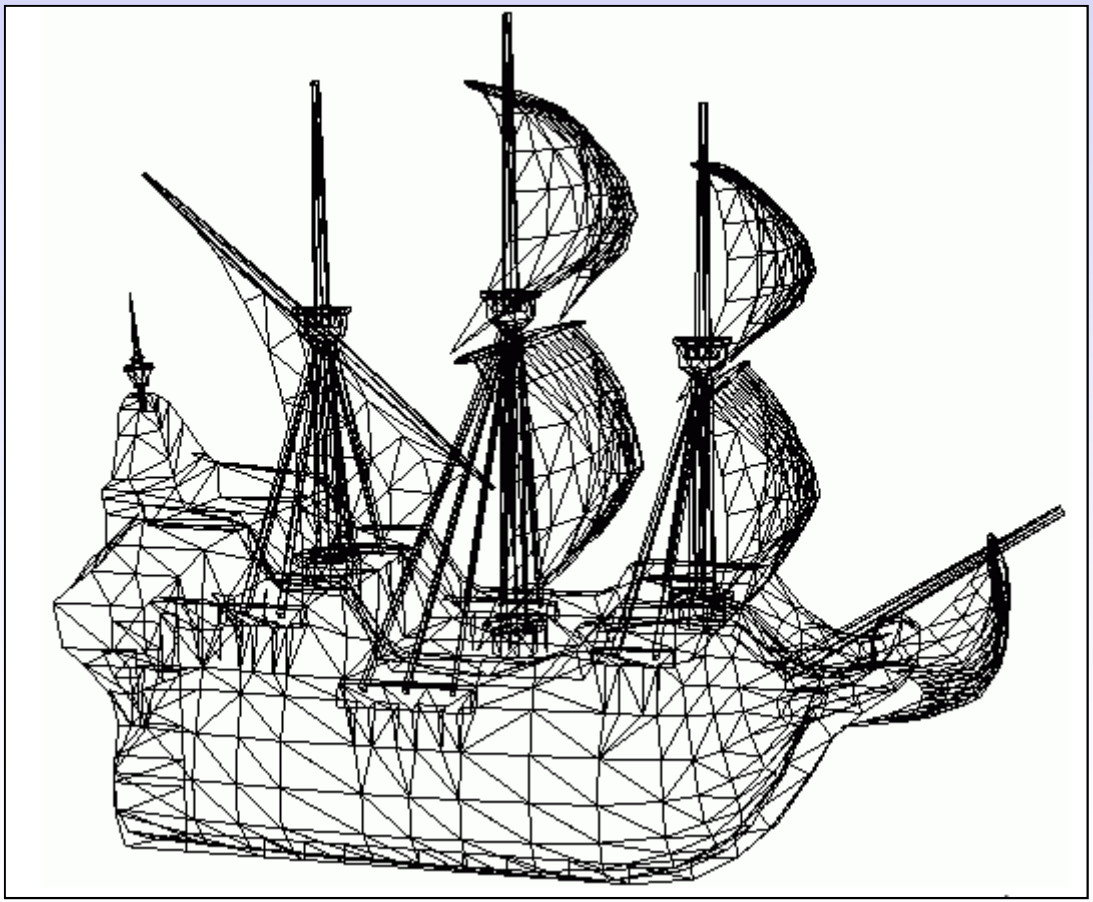
From simple to complex…



Shapes Built From Polygons

Smooth surfaces approximated by lots of flat triangles…

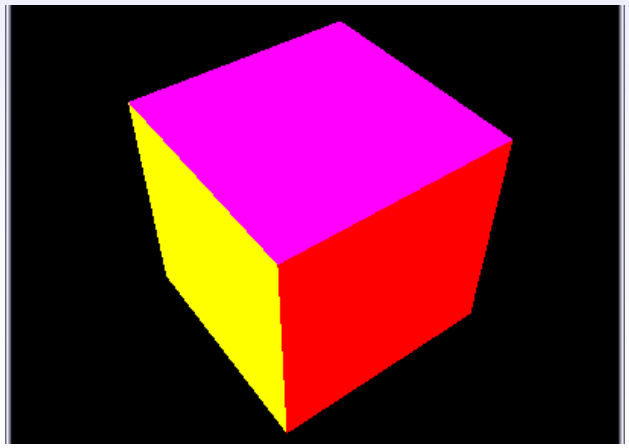




Polygons

3D objects can be built from 2D polygons. A polygon is a closed planar (flat) figure with 3 or more sides.

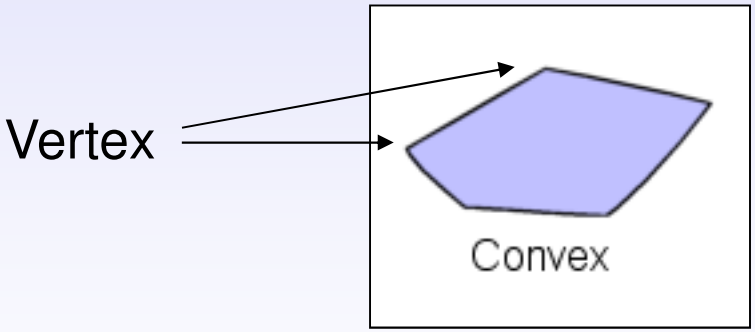
- e.g. a triangle, square or pentagon, however it does not have to be regular
- We build a 3D cube from 2D square polygons, a pyramid from triangle polygons
- The term for a 3D shape built from flat 2D polygons is a **polyhedron**



Polygons (2)

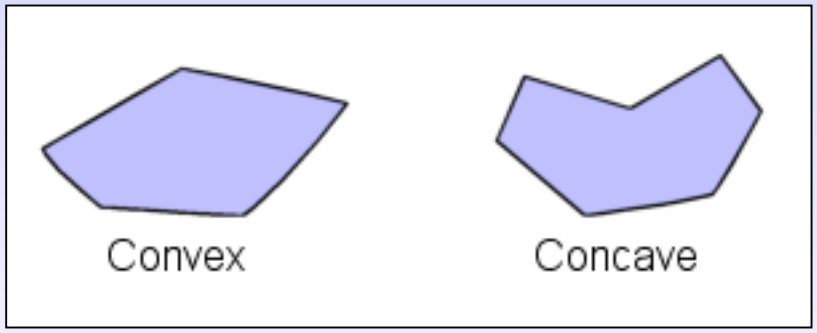
A polygon is defined by its set of 'corners' which are known as its vertices (the singular is vertex).

- Each vertex is a 3D point.
- The shape bounded by the set of vertices is filled to show a planar area in the 3D space - a polygon.



Polygons (3)

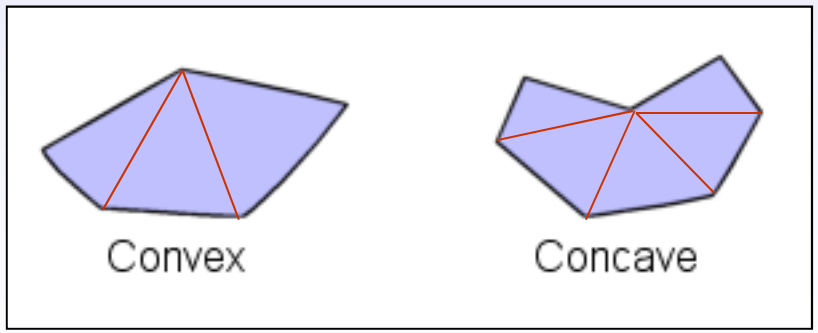
You can build lots of polygon shapes, however we try to make sure that our polygons and polyhedrons are convex rather than concave.

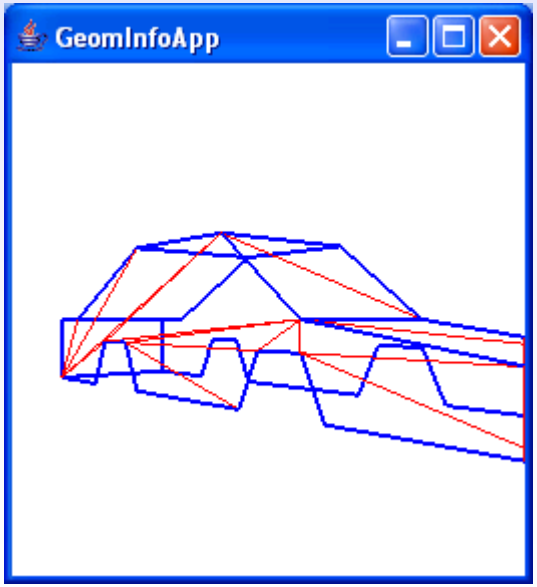


- One difficulty with concave shapes is that they are not easy to fill quickly. Try thinking about writing a fill algorithm for a concave versus a convex shape.

Polygons in Java 3D

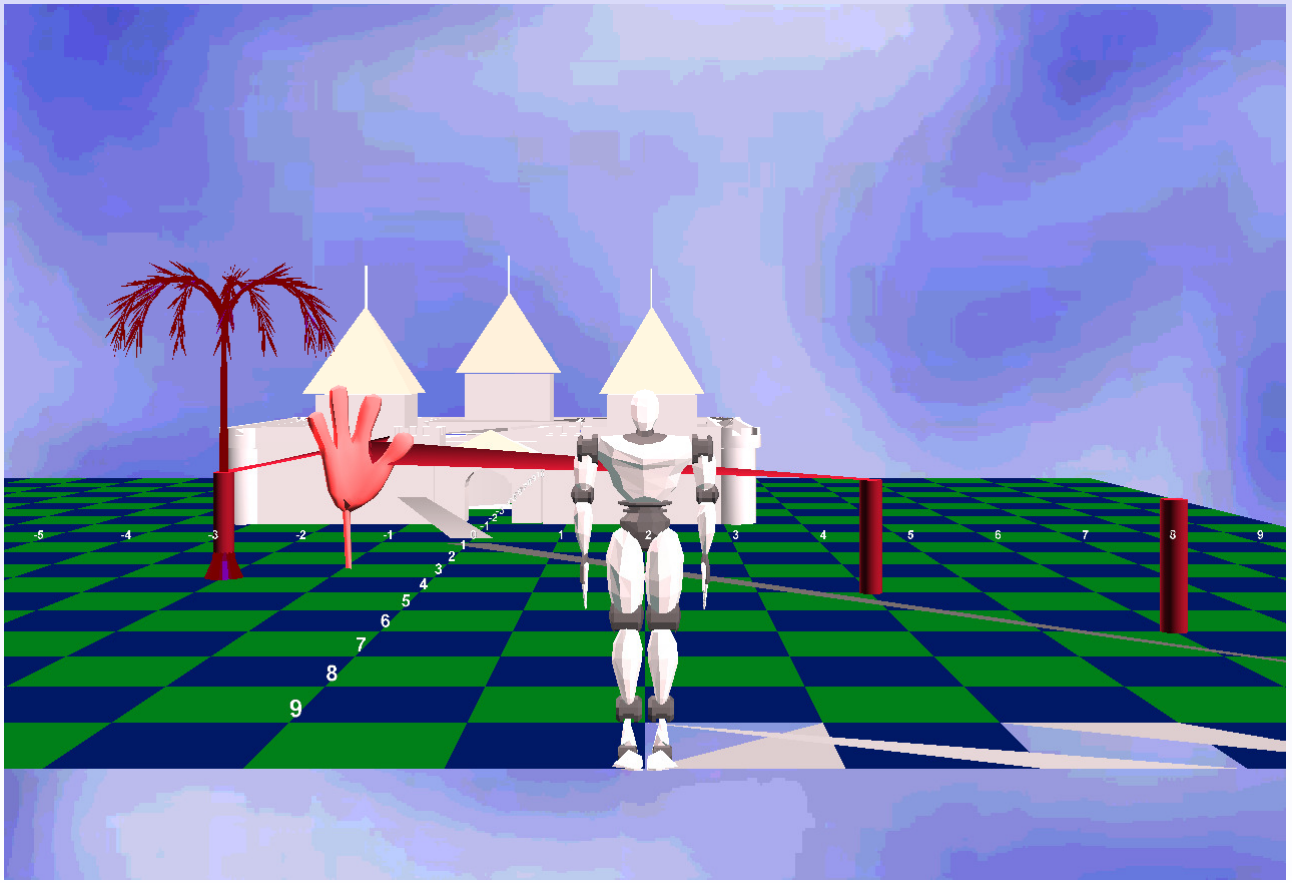
- Java 3D will only render polygons with at most 4 sides (quadrilaterals)
- Complex polygons must be subdivided, usually into triangles
  - Responsibility of the programmer
  - BUT methods supplied to do this for you
  - Tessellation
  - Triangulator utility class





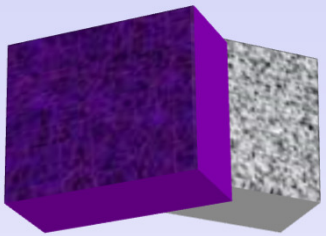
Rendering a 3D Scene

Do we need to render all the polygons in the scene?



Hidden Surface Removal

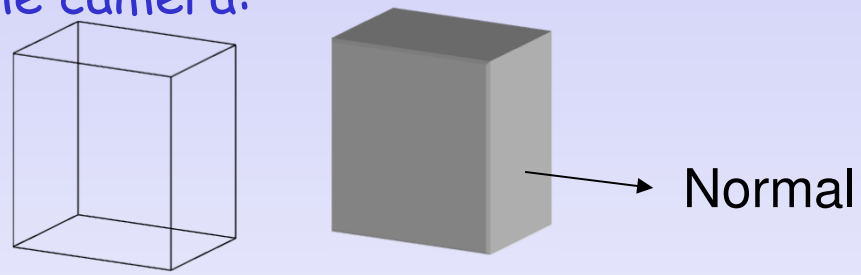
Take the situation where we have two polyhedrons, one on top of the other:



- We would like to keep the number of polygons we draw to the minimum necessary.
  - All of the back facing polygons are hidden
  - One of the forward facing polygons in the grey cube is hidden
  - There are 12 (2x6) polygons we could draw and fill but only 5 are actually visible
  - We also need to draw them in the right order...

Back Face Culling

In this approach, we only draw the polygons in an object that we know are facing the camera:

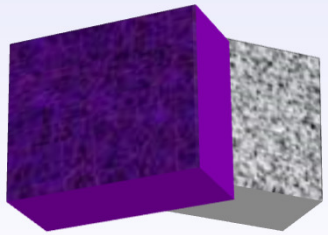


- Each polygon has a 'normal' - a vector that is at right angles to the face of the polygon.
- If the angle between the camera and the normal is less than 90 degrees, the polygon is facing the camera.
- All the polygons on the back of the above cube have a normal that faces away from the camera i.e their normal is greater than 90 degrees.
- We can calculate this angle using the 'dot product' of two angles, see DGJ, p377 for more details.

Draw Order - Painters Algorithm

You are probably thinking - we still don't know what order to draw things in…

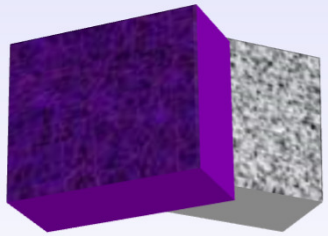
- The simplest approach is to use the "Painter's Algorithm"
- Draw all the polygons from the furthest away through to the nearest.
- The nearest polygons draw over the furthest
- It is somewhat inefficient since we overdraw many times



Reverse Painters Algorithm

A better variation on this theme is to use the "Reverse Painters Algorithm"

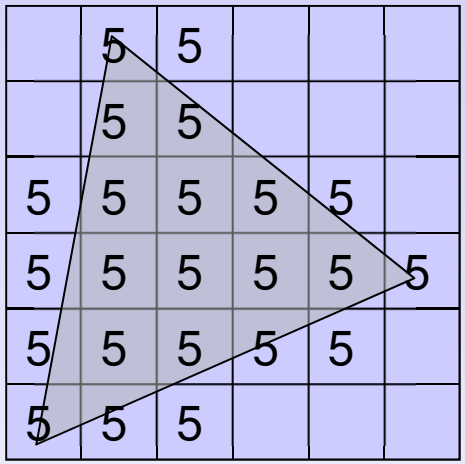
- We draw the nearest polygons first
- Subsequent draw commands which try to fill a pixel that has already been drawn are ignored since we must now be behind a front facing polygon

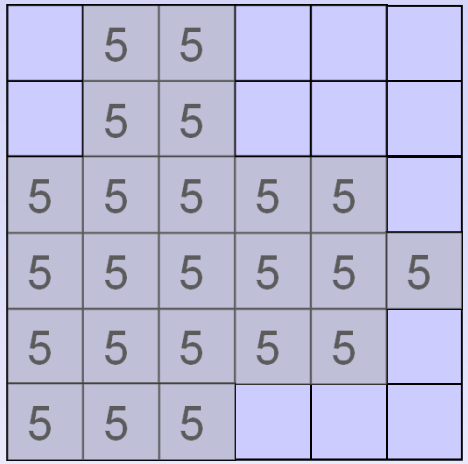


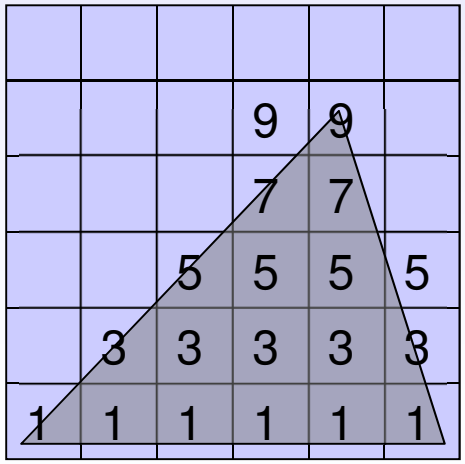
Z-Buffering

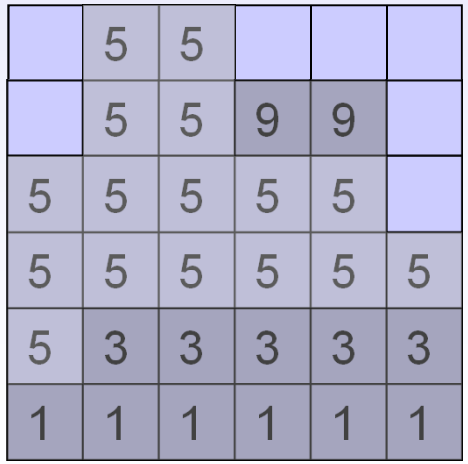
- A buffer is created the same size as the screen (called the Z-buffer because it stores information about the Z or depth access).
- Each pixel in a polygon is assigned a depth based on its z co-ordinate
- An attempt is then made to draw each pixel in the polygon onto the Z-buffer
- If the value in the Z buffer is greater than the one allocated to the pixel in the polygon, the pixel is drawn and its value becomes the new value in the Z buffer
- This allows slanting and overlapping polygons to drawn correctly
- A good diagram illustrating this is shown in DGJ, p485…

Z-Buffering (2)









The End