Section B

1.

1. Polygon soup: a huge list of individual polygons coloured and drawn in order.  
   Undesirable because:  
   A- waste of space  
   B- brute force rendering and makes interaction with the model complex  
   C- loss of semantics (is this polygon a cow’s or a table?)
2. Vertex list. Stores the coordinates   
   Edge List indexing into the vertex list.  
   Face list indexing into the edge list
3. Start a vertex with   
   V x-cord y-cord z-cord  
   Start an edge with  
   E vertex1 vertex2  
   Start a face with  
   F edge1 edge2 edge3 ...
4. Scan conversion: the technique of using a true geometry of a line and approximating it using the nearest pixels available.  
   Using Sweep-line algorithm:  
   Step down a pair of edges.  
   Go down scanline by scanline.  
   Find start and end of the part of the scanline inside the triangle.  
   Fill the pixels inside the triangle for that scanline.
5. When we are scan-conversion an object, whenever we generate a pixel P we check that some other object nearer to the eye, also maps to P.
6. because they are floating point algorithm they tend to lose information when rounding.   
   By using z-buffer which stores the depth of the surface

2.

1. Local: We treat each object independent of any other object.  
   Global: We treat all object together, and model their interaction and effects they cause on each other.
2. Illumination = Ambient illumination + diffuse reflectivity + specular reflectivity taking into account distance.  
   I = kac iac + (ipc/ d’)[kdc (N.L) + Ks (R.V)n]   
   1. Ambient illumination: the general level of illumination caused by multiple reflection in a scene.   
      Ambient illumination = ka \* iaWhere ka The ambient reflection coefficient, number between 0 and 1  
      Ia the intensity of the ambient light (constant)   
      Effect: each object is uniformly illuminated and we lose 3d information.
   2. Diffuse effects: When a light gets reflected by a surface we describe the diffuse reflector using kdIdiffuse = ipkdcos(theta)  
      Where ip the intensity of light source.   
      kd - the diffuse reflection coefficient. This is between 0 and 1.  
      And theta is the angle between the surface normal (N) and direction of light source (L)  
      Hence   
      Diffuse reflector = ip \* kd (N.L)  
      Taking into account the distance of the light   
      Diffuse reflector = ip/d’ \* kd(N.L)  
      Where d’ = kc +kl + kqd2   
      Kc is the constant lighting distance coefficient.  
      Kl is the linear lighting distance coefficient.  
      Kq is the quadratic lighting distance coefficient
   3. Specular effects: modelling the effect of reflected light in relation to the observer’s position.   
      iSpecular = ks\* ip/d’ \* (R.V)  
      Where R is the vector giving the direction of maximum specular reflection  
      V is the vector pointing to the observer’s position  
      And ks is an approximation of inbound angle and wavelength, with it being a number between 0 and 1
   4. Coloured lights and surface:   
      We represent that by saying iac where c refer to R/G/B (red, green or blue)
   5. Erm what
3. We use it in Gouraud algorithm to compute the colour.  
   So we compute the 3 pixels that form a triangle, Ca, Cb, Cc  
   For each scanline  
   We average the colour between Ca and Cc   
   Average between Cb and Cc  
   And the average between the average resulted from above
4. Problems:  
   - sometime the edges are shaded away: solution: tag the edges in the data structure  
   - mach banding may still be visible   
   - specular highlights may be distorted or averaged away   
   Can be overcome using Phong interpolation, instead of interpolating intensity we’re now interpolating normal vectors.  
   Compute vertex normal at each vertex, then:  
   For each scanline  
    Average Nleft from Na and Nc  
    Average NRight from Nb and Nc  
    Average between Nleft and Nright across the scanline, computing the colour of each pixel.

---------------------------------------------------------------------------------------------------------------------------

Section C

3-

1. Connected component labelling: aims to identify pixels that are connected in an image by checking the pixel value.   
   There are two types of connectivity:   
   4 connected - objects joining at corners can be disconnected   
   And 8 connected: objects join on corners but might pierce thin objects.  
   Regardless of connectivity we choose:  
   We scan the image top to bottom, left to right, checking for adjacent pixels (adjacency is based on the type of connectivity we used), there are 3 cases here:  
   Case 1: if adjacent pixels aren’t of the same value, then we give the current pixel the next free label.  
   Case 2: if more than one pixel of the same the value, we give the current one the minimum out of them, and record their equivalency in the equivalent table.  
   Case 3: if one pixel is the same value, we assign to it the same label.  
   After we can the whole image, we do another pass:  
   Working from left to right, we assign to each label the label assigned to it in the equivalent table.
2. Property is the grey and colour values.   
   i) Threshold the grey values of an image by simple equation:  
   If image.value > threshold = 1; else = 0.   
   After that we have a binary image where 1 means it is an object and 0 is a background and work on that. So if two pixels have value 1 and they are adjacent they are most likely connected.   
   ii ) covariance colours in a coloured images.
3. Algorithm 1:   
   Using moments of area:  
   Mab =   
   Where x’, y’ are the centre of gravity.   
   Orientation is given by   
   Tan2θ = 2M11/(M20 - M02)   
   And (M10/M00, M01/M00) gives region’s centre of gravity.  
   If a = 1 and b = 0, then it will give the sum of x values of region’s pixels  
   Useful to compute position, orientation and size of region. And works on non-binary images. Can use it to discriminate blobs based on size or shape.  
     
   Algorithm 2:  
   Chain codes:  
   We trace along the object outline following pixels on boundary.   
    With 0 going forward, 2 right, 4 back and 6 left. Then 1, 3, 5, and 7 in between.  
   It is position independent but orientation dependent.  
   The differntial chain code is orientation independent by just using 0 as the direction of last movement.  
   Can derive area, perimeter and many other measurements from the chain codes.  
   The perimeter is: length 1 \* pixel if an even code, sqrt(2) \* pixel if an odd code, then the sum of all the boundry gives the perimeter.  
   The area is: the sum of codes where:  
   0 = 0, 1 = h + 1/2, 2 = h, 3 = h-1/2, 4 =0 , 5 = -h - 1/2, 6 = -h, 7 = -h+1/2  
   Where h is the measurement from arbitrary datum.
4. i) Will first give it a predefined gestures. Such as pointing.  
   Then will do CCA on it and calculate the area, perimeter and colour distribution of pointing gesture.   
   The more gestures we have the more training dataset we need.  
     
   ii) will evaluate it by running it on a confusion matrix and cross validate it that each gesture accurately get read. THE FK I KNOW PLEASE I HATE THIS MODULE

4) fkme