

1.(a) — bookwork

- A BSP-tree (binary space partition tree) for a set of (convex, simple, flat) polygons is a binary tree where each node splits a set of polygons into a set in front and a set behind a plane. The plane is typically determined by one of the polygons, which is stored at the corresponding node. Polygons intersecting with this plane are usually cut into two. [3]
- Construction principle for polygons in 3D:
 - (i) choose a polygon and use it to define a partition plane [1]
 - (ii) Split the remaining set of polygons into two sets of polygons in front of and behind the plane. [1]
 - (iii) If a polygon intersects the plane cut it along the plane into two polygons and sort them in the same way. [1]
 - (iv) Recursively continue splitting the polygon sets until the set consists of at most one polygon. [1]

Total: [7]

1. (b) — bookwork

- For last edge set $v_m \leftarrow v_0$; initialise intersection index $k \leftarrow 0$ and intersection point array p [1]
- Determine half-space of first vertex: $s \leftarrow n^t v_0 - d$ [1]
- for all other vertices: $l \leftarrow 1$ to m [1]
 - Determine half-space of l -th vertex: $r \leftarrow n^t v_l - d$ [1]
 - if $s = 0$ (first vertex is intersection), $p[k + +] \leftarrow v_{l-1}$; [1]
 - else if $sr < 0$ (there is an intersection),
 - Intersect plane $n^t x - d = 0$ with parametric line segment $t \mapsto v_{l-1} + t(v_l - v_{l-1}), t \in [0, 1]$ (there must be an intersection):
 - $t \leftarrow (d - n^t v_{l-1}) / (n^t (v_l - v_{l-1}))$
 - $x \leftarrow v_{l-1} + t(v_l - v_{l-1})$
 - $p[k + +] \leftarrow x$ [5]
 - $s \leftarrow r$ [1]
- return k intersection points from p [1]

Total: [12]

1. (c) — unseen

- Assume the plane splits the polygon into more than two polygons. Then there must be two polygons on the same side of the plane. Pick one point from each of these two polygons. The line between these points cannot be completely inside the two polygons, which means the original polygon is not convex.
 (can also be done by showing that there are at most two intersection points based on 1b as otherwise there would be more than one line segment in the intersection and so the original polygon cannot be convex). [4]
- Trivially the result is convex if the plane did not split the polygon. Otherwise, the resulting two polygons are convex: given two points in one of the polygons, the line segment between them cannot cross the splitting plane by construction and the original polygon was convex. [2]

Total: [6]

2.(a) — bookwork

- Ambient light: comes from all directions and is reflected in all directions; intensity the same everywhere [3]
- Diffuse light: comes from a specific direction and is reflected in all directions; intensity depends on the angle between the incoming light and the surface normal [3]
- Specular light: comes from a specific direction and is reflected in a preferred direction; intensity depends on the angle between the direction of the perfectly reflected light and the direction to the viewer [3]

Total: [9]

2. (b) — bookwork

- Ambient: global ambient light

$$I(p, L) = L$$

[1]

- Directional: treated as being infinitely far away and hence the light comes from a single direction d and no position:

$$I(p, L) = L$$

[2]

- Point: light emitted radially from a single point in all directions. Defined by a position l and constant, linear and quadratic attenuation k_c, k_l, k_q to reduce light intensity with distance from the light source:

$$I(p, L) = \frac{L}{k_c + k_l \|p - l\| + k_q \|p - l\|^2}$$

[3]

- Spot: similar to point light source, but light emitted in a cone from a single point. Defined as point light source with additional unit cone direction s , and spot cut-off exponent τ which determines the drop of intensity as the angle between s and the direction to p increases:

$$I(p, L) = \frac{(s^t((p - l)/\|p - l\|))^\tau L}{k_c + k_l \|p - l\| + k_q \|p - l\|^2}$$

[3]

Total: [9]

2. (c) — unseen

- Simulation: render a second shadow polygon for each polygon, each light source, and each polygon which lies in the shadow [3]
- A shadow polygon can be computed by a perspective projection transformation where the centre of projection is the point light source [2]
- Suitable for simple scenes, e.g. shadows on large terrain polygons [1]
- No longer practical when objects cast shadows on other objects due to large amount of computations [1]
(alternative answers, e.g. relating to ray tracing or shadow maps, possible; basic principle with indication of how to compute it and suitability/limitations should be discussed)

Total: [7]

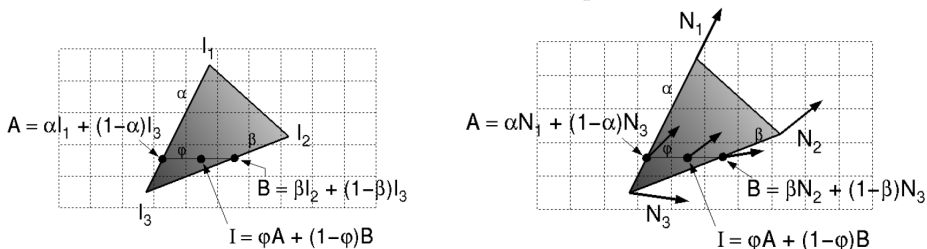
3. (a) — bookwork

- sort edges by maximum y coordinate [1]
- create empty active edge list [1]
- for each scan-line $s =$ top to bottom [1]
 - insert edges into active edge list if their maximum y coordinate is s [1]
 - remove edges from active edge list if their minimum y coordinate is larger than s [1]
 - update x coordinate of every active edge (DDA, Bresenham, ...) [1]
 - sort active edges by current x coordinate [1]
 - for each pair (x_{2k}, x_{2k+1}) of active edges (left-to-right, every x coordinate only used once; when scan-line goes through a vertex of the polygon: vertices at extrema- y direction of both edges is either up or down—have to be considered twice), draw span from x_{2k} to x_{2k+1} in scan-line s in single colour [3]

Total: [10]

3. (b) — bookwork

- Gouraud shading: Evaluate illumination model for each vertex and bilinearly interpolate the results to get RGB intensities for each pixel. [2]
- Phong shading: Bilinearly interpolate the vertex normals for each pixel and evaluate the illumination model for each pixel with interpolated normals to get intensities (for red, green, blue). [2]
- Bilinear interpolation used in both techniques where interpolation parameters are available from the scan conversion process:



(diagrams not required, formulae are sufficient with indication of what the symbols are) [4]

Total: [8]

3. (c) — unseen

- Construct a BSP tree based on the edges of the area. This partitions the 2D plane into polygonal areas lying fully inside or fully outside of the area by recursively partitioning 2D areas into two sub-areas along the edges. The areas lying fully inside form the convex polygons for rendering. [4]
- At the root the BSP tree partitions the 2D plane into two half-spaces which are both convex. Given a convex area, a node partitions it into two sub-spaces along a line. The two sub-spaces are convex (otherwise the original space is not convex). So the polygonal areas at the leaves of the BSP tree are convex. (other tessellation schemes with similar justification are also possible) [3]

Total: [7]

4. (a) — bookwork

- Radiosity models light as energy transfer between small surface patches based on thermal heat transfer, each surface can emit as well as absorb energy. Radiosity is the rate at which energy leaves a surface patch and essentially describes its brightness / intensity. It combines both the light emitted by the surface as well as the light reflected at the surface. Depending on the relative arrangement between two surface patches a certain fraction of the radiosity of one patch reaches the other patch. [4]
- Radiosity equation:

$$B_k = E_k + \rho_k \sum_j B_j F_{kj} \text{ for } k = 1, \dots, n$$

- B_k : radiosity/brightness of patch k [3]
- E_k : light/energy emitted by patch k [1]
- ρ_k : fraction of incoming light/energy reflected by patch k [1]
- F_{kj} : form factor between surface k and j indicating the fraction of light / energy leaving surface j and reaching surface k . [2]

Total: [12]

4. (b) — bookwork

- Convert the parametric surfaces into a mesh consisting of small patches. [1]
- Compute the form factors F_{kj} for all patch pairs. [1]
- Set the emitted light E_k of each patch according to its material attributes. [1]
- Compute the energy intensity transfer by (approximately) solving the linear radiosity equation for red, green, and blue light. [1]
- Use the solutions B_k to shade the patches. [1]

Total: [5]

4. (c) — unseen

- The radiosity algorithm works with patches and each patch is an emitter as well as a reflector. Surface patches are usually too big to represent a point and hence in general radiosity cannot handle point light sources well. [2]
- A point light source would be represented by a small patch of approximately the size of only a few pixels which emits a high level of energy. This is likely to cause artefacts due to numerical problems computing the form factors for such a small patch. [3]
- To deal with such light sources one could combine radiosity with a view-point dependent ray tracing algorithm which traces rays from all visible pixels back to the point light source and adjusts the brightness levels accordingly. [3]

Total: [8]