

G-II Exercises

1. Polygon Orientation

- (a) Given n three-dimensional points p_1, \dots, p_n in this sequence, what is the standard convention in OpenGL to determine if a viewer looks at the front or the back side of the polygon?
- (b) Assume the unit vector v points towards the viewer from point p_1 . Derive the formulas to determine whether the viewer is looking at the front or the back of the flat, simple polygon.

2. Half-Edge Representation

The surface of arbitrary manifold solid objects can be approximated by a set of small polygons. Devise a boundary representation data structure to represent such an approximation. The data structure should allow for efficient queries of the type:

- Which vertices bound this edge?
- On which polygons does this edge lie?
- Which polygons are adjacent to this polygon?
- Which edges are adjacent to this vertex?

Also provide simple code fragments that answer these questions. What is the time complexity of these fragments? Hint: you may, as the title suggest, consider how to store the edges and their connectivity using half-edges.

3. Monochromatic Phong Illumination

Assume you have a single, monochromatic light source at position p_l emitting ambient light of intensity I_a , diffuse light of intensity I_d and specular light of intensity I_s with shininess exponent α . The intensity of the diffuse and specular light that travelled a distance d from the light source is $1/(a_0 + a_1d + a_2d^2)$ of the original intensity.

Compute the intensity of the light reaching a viewer at position p_v from a surface point p_s with normal n_s using the Phong illumination model. Assume that the surface reflects R_a of the incoming ambient light, R_d of the incoming diffuse light and R_s of the incoming specular light. The computation should only use the parameters listed here and provide formulas for any other scalars or vectors needed.

4. Transformations

- (a) Explain how to convert standard 3D Cartesian coordinates (x, y, z) to homogeneous coordinates and how to convert homogeneous coordinates to standard 3D Cartesian coordinates.
- (b) Show how to perform a 2D rotation about an arbitrary point. Provide a matrix in homogeneous coordinates for each step in the operation.
- (c) Show how to perform a 3D rotation about an arbitrary axis. Again, give matrices in homogeneous coordinates for each step in the operation.

G-II Exercise Solutions

1. Polygon Orientation

- (a)
 - we look at the front side of the polygon if the sequence of points is counter-clockwise on the display
- (b)
 - choose three non collinear points p_l, p_k, p_m with $l < k < m$
 - Let $v_1 = p_k - p_l$ and $v_2 = p_m - p_k$
 - Let $n = v_1 \times v_2$
 - $n^t v < 0$: front side
 - $n^t v > 0$: back side
 - $n^t v = 0$: neither
 - (Signs reversed if $n = v_2 \times v_1$, or point sequence is $m < k < l$)

2. Half-Edge Representation

We use half-edges to represent the connectivity between faces and vertices. To represent general manifold objects we also need **Body**, **Lump** and **Shell** similar to the general boundary representation. A shell represents a connected surface mesh build from polygons connected via half-edge pairs. Instead of storing the edges, we store only the two half-edges. So for each polygon (the **Facet** class), we store a reference to an arbitrary half-edge on its boundary. The half-edge contains a pointer to the next half-edge in sequence around the polygon, which gives the complete boundary loop (traversed in anti-clockwise order when we look at the surface from outside to get the orientation and the normal of the facet). The half-edge also contains a reference to the end-vertex of the half-edge with respect to its orientation. A reference to its partner half-edge lying on the adjacent polygon provides the polygon adjacency information and the other vertex of the edge. As we have only planes and straight lines, we just have to store the Vertex geometry.

```

class Body {
    Lump *lumps;    // first in list of lumps
};

class Lump {
    Lump *next;    // next in list of lumps (circular link)
    Shell *shells; // first in list of shells
    Body *body;    // pointer back to body (simpler for bottom-up traversal)
};

class Shell {
    Shell *next;    // next in list of shells (circular link)
    Lump *lump;     // pointer back to lump
    Facet *facet;   // an arbitrary facet of the surface mesh
};

class Facet {
    HalfEdge *edge; // one of the half-edges on the boundary of the facet
};

class HalfEdge {
    Vertex *vertex; // vertex at the end of the oriented half-edge
    Edge *partner;  // other half-edge with opposite orientation
    Facet *facet;   // the polygon on which this half-edge lies
    HalfEdge *next; // next half-edge around the polygon
};

```

```
};

class Vertex {
    float x, y, z;    // vertex coordinates
    HalfEdge *edge;  // one of the half-edges containing this vertex as end-vertex
}

```

Alternatives are possible, but this is most efficient for the queries below. Note that we could store the vertices in an array and use array indices.

Code fragments to answer the following questions:

- Which vertices bound this edge?
An edge is represented by a half-edge and we get its vertices via its end-vertex and its partner's end vertex. No need to check for NULL pointers if we have a manifold mesh (otherwise we have to).

```
Vertex *vertex1 = edge->vertex;
Vertex *vertex2 = edge->partner->vertex;

```

Time: $O(1)$ (constant number of operations)

- On which polygons does this edge lie?
This works in the same way than above, just we get the facets instead of the vertices.

```
Facet *facet1 = edge->facet;
Facet *facet2 = edge->partner->facet;

```

Time: $O(1)$ (constant number of operations)

- Which polygons are adjacent to this polygon?
We get the half-edge stored at the facet and can move around the facet by following the next pointers until we are back at the original half-edge. For each of the half-edges we store the facet of its partner. No need to check for NULL pointers if we have a manifold mesh.

```
List<Facet> adjacent;
Edge *edge = facet->edge;
do {
    adjacent.push (edge->partner->facet);
    edge = edge->next;
} while (edge != facet->edge);

```

Time: $O(k)$ where k is the maximum number of edges bounding a single facet in the mesh. So the time depends linearly on k . Usually k is small and bound by a constant (e.g. 3 for triangular meshes), so this can be treated as constant time as well.

- Which edges are adjacent to this vertex?
Similar to above. From the vertex we can find one half-edge. Moving to the next half-edge in the polygon and taking the partner of this half-edge gives the next half-edge which contains this vertex (note that we could store any of the two half-edge partners). We continue this until we are back at the original half-edge.

```
List<HalfEdge> adjacent;
HalfEdge *edge = vertex->edge;
do {
    adjacent.push(edge);
    edge = edge->next->partner;
} while (edge != vertex->edge);

```

Time: $O(l)$ where l is the maximum number of edges adjacent to a single vertex in the mesh. So the time depends linearly in l . Often l is also bound by a small constant, so this can be treated as constant time as well.

Note, the radial-edge and a modified winged-edge data structures are capable of representing general, non-manifold meshes.

3. Monochromatic Phong Illumination

- Let $l = (p_l - p_s) / \|p_l - p_s\|$
- Let $v = (p_v - p_s) / \|p_v - p_s\|$
- Ambient light reflected by the surface:

$$L_{\text{ambient}} = R_a I_a.$$

- Diffuse light reflected by the surface:

$$L_{\text{diffuse}} = R_d l^t n_s I_d$$

- Specular light reflected by the surface considering viewing direction:

$$L_{\text{specular}} = \begin{cases} R_s (r^t v)^\alpha I_s & \text{if } v^t r > 0 \\ 0 & \text{otherwise} \end{cases}$$

where r is the perfect reflection of l :

$$r + l = 2(l^t n_s) n_s$$

$$r = 2(l^t n_s) n_s - l$$

- Radial attenuation of specular and diffuse light:

$$f_{\text{radatten}} = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

with

$$d = \|p_l - p_s\| + \|p_v - p_s\|$$

- Visibility: light source behind object?

$$V = \begin{cases} 0 & \text{if } l^t n_s \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

- Total:

$$L_{\text{total}} = L_{\text{ambient}} + V f_{\text{radatten}} (L_{\text{diffuse}} + L_{\text{specular}})$$

4. Transformations

(a) Conversion

- to homogeneous coordinates:

$$(x, y, z) \rightarrow (x, y, z, 1) \text{ or } (x, y, z) \rightarrow (xw, yw, zw, w) \text{ for } w \neq 0$$

- from homogeneous coordinates:

$$(x, y, z, w) \rightarrow (x/w, y/w, z/w)$$

(b) Rotate by ϕ about $p = (x, y)$:

- (1) Translate p to the origin:

$$A = \begin{bmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{bmatrix}$$

- (2) Rotate by ϕ about origin:

$$B = \begin{bmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- (3) Translate origin to p :

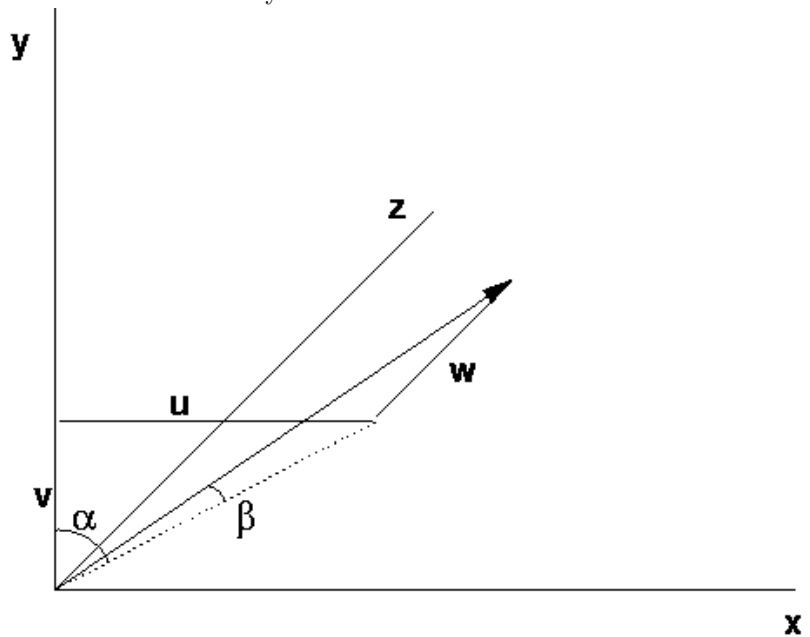
$$C = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

- (c) Rotate by ϕ about an axis passing through $p = (x, y, z)$ and pointing in direction $d = (u, v, w)$:

- (1) Translate p to the origin

$$\begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- (2) Rotate the axis onto the z -axis by:



- (2a) rotating the axis about the z -axis into the yz -plane

$$\begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with

$$\alpha = -\arctan\left(\frac{u}{v}\right)$$

- (2b) rotating the result about the x -axis onto the z -axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta & 0 \\ 0 & \sin \beta & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with

$$\beta = -\arctan\left(\frac{\sqrt{u^2 + v^2}}{w}\right)$$

- (3) Rotate by ϕ about the z -axis

$$\begin{bmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- (4) Reverse the rotation (2b) by replacing β with $-\beta$ and then reverse the rotation (2a) by replacing α with $-\alpha$
- (5) Reverse the translation (1) by replacing the minus signs with plus signs