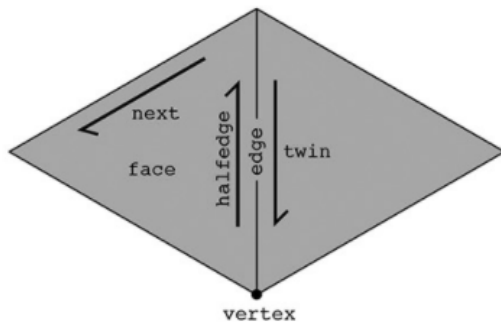


HALF-EDGES AND RAY TRACING 5

CS 184: FOUNDATIONS OF COMPUTER GRAPHICS

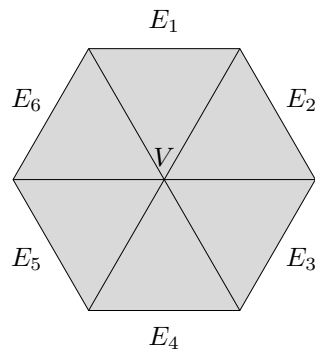
1 The Half-edge Data Structure

The half-edge data structure is a powerful representation that enables us to navigate a polygon mesh and perform various operations. The mesh elements have the following relationships:



Data structure	Attributes
Vertex	one halfedge
Edge	one halfedge
Face	one halfedge
Halfedge	twin, next, vertex, edge, face

1. Starting from a given vertex, traverse the mesh and return a `std::vector` containing all of the edges that are opposite that vertex. In the diagram below, E_1, \dots, E_6 are the edges opposite V . **Hint:** Start with the vertex's halfedge, then perform a do-while loop until we return to the original halfedge.



```
std::vector<EdgeIter> getOppositeEdges(VertexIter v) {
```

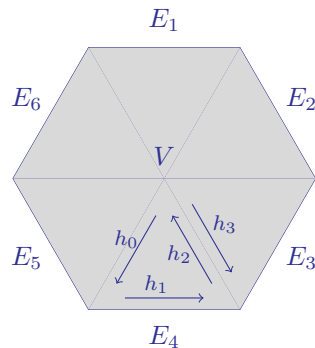
Solution:

```
{  
    std::vector<EdgeIter> edges;  
    HalfedgeIter h = v->halfedge();  
    HalfedgeIter start_h = h;  
    do {  
        edges.push_back(h->next()->edge());  
        h = h->next()->next()->twin();  
    } while (h != start_h);  
}
```

```

} while (h != start_h);
return edges;
}

```



As justification, assume without loss of generality that $V \rightarrow \text{halfedge}() = h_0$ is as shown in the figure above. The half-edges are oriented counter-clockwise, so the figure above shows $h_1 = h_0 \rightarrow \text{next}()$, $h_2 = h_1 \rightarrow \text{next}()$, and $h_3 = h_2 \rightarrow \text{twin}()$.

In this example, the first iteration of the loop pushes $E_4 = h_0 \rightarrow \text{next}() \rightarrow \text{edge}()$ to edges, and h moves from h_0 to h_3 . We iterate until h eventually returns to h_0 , at which point $\text{edges} = [E_4, E_3, E_2, E_1, E_6, E_5]$.

There is nothing special about this choice of h_0 ; the same algorithm would work if we had started with any other half-edge with vertex V .

Note: In this solution, we traverse the triangles in counterclockwise order. It is also valid to traverse in clockwise order via $h = h \rightarrow \text{twin}() \rightarrow \text{next}()$.

- Let's write a function that locally attenuates noise in a mesh. Given a vertex v at position x , let

$$L(v) = \frac{1}{n} \sum_{v_j \in N(v)} x_j - x,$$

where $N(v)$ is the set of neighboring vertices of vertex v , x_j is the position of neighboring vertex v_j , and n is the number of neighboring vertices.

Apply $L(v)$ to slightly move v from position x to a new position x' , making the mesh slightly smoother around v : $x' = x + kL(v)$, where k is a weight. We call this a diffuse operation on v 's position.

```
void diffuse(VertexIter v, float k) {
```

Solution:

```

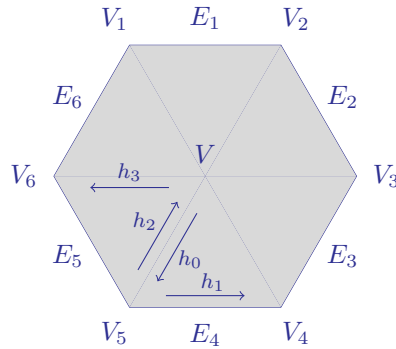
{
    Vector3D L(0, 0, 0);
    HalfedgeIter h = v->halfedge();
    HalfedgeIter start_h = h;
    int n = 0;
    do {
        VertexIter v_j = h->next()->vertex();
        Vector3D dir = v_j->position() - v->position();

```

```

L = L + dir;
n++;
h = h->twin()->next();
} while (h != start_h);
v->position() = v->position() + k * L / n;
}

```

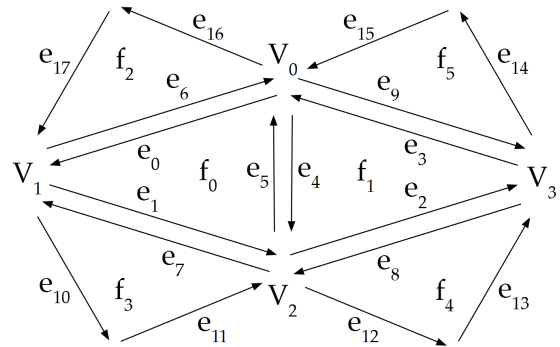


Following the same setup as in the previous part, assume without loss of generality that $V \rightarrow \text{halfedge}() = h_0$ is as shown in the figure above. The half-edges are oriented counter-clockwise, so the figure above shows $h_1 = h_0 \rightarrow \text{next}()$, $h_2 = h_0 \rightarrow \text{twin}()$, $h_3 = h_2 \rightarrow \text{next}()$. In particular, $V_5 = h_0 \rightarrow \text{vertex}()$. The first iteration of the loop adds $(V_5 \rightarrow \text{position}()) - (V \rightarrow \text{position}())$ to dir , and h moves from h_0 to h_3 .

We iterate until h eventually returns to h_0 , at which point we have processed $[V_5, V_6, V_1, V_2, V_3, V_4]$ in order.

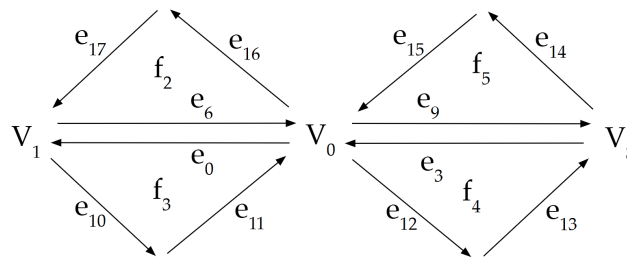
3. Fill in the blanks so that the resulting procedure collapses the edge connecting vertices V_0 and V_2 .

- (i) $e_{11} \rightarrow \text{next}() =$ _____
- (ii) $f_3 \rightarrow \text{halfedge}() =$ _____
- (iii) $V_0 \rightarrow \text{halfedge}() =$ _____
- (iv) $e_3 \rightarrow \text{face}() =$ _____
- (v) $e_{12} \rightarrow \text{vertex}() =$ _____
- (vi) $V_1 \rightarrow \text{halfedge}() =$ _____
- (vii) _____
- (viii) _____
- (ix) _____
- (x) _____
- (xi) _____
- (xii) _____



- (xiii) Delete vertex V_2
- (xiv) Delete half-edges $e_1, e_2, e_4, e_5, e_7, e_8$
- (xv) Delete faces f_0, f_1

Solution: Based on the half-edges deleted in step (xiv), we can construct the desired output:

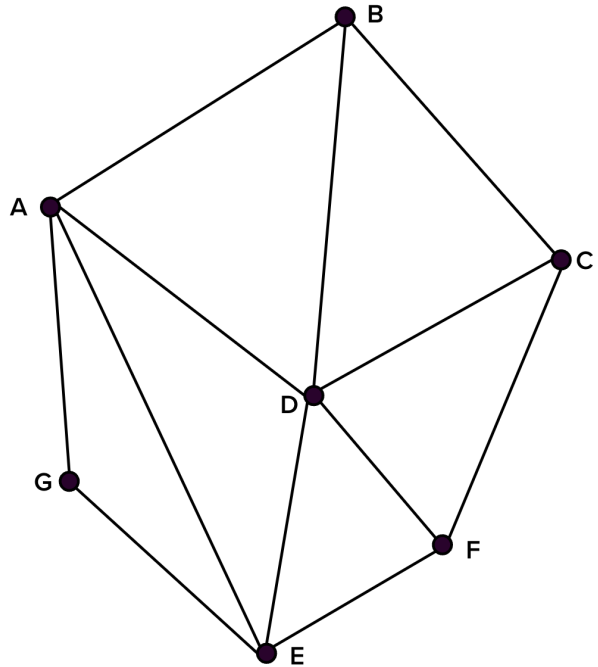


- (i) $e_{11} \rightarrow \text{next}() = e_0$
- (ii) $f_3 \rightarrow \text{halfedge}() = \text{any of } e_0, e_{10}, e_{11}$
- (iii) $V_0 \rightarrow \text{halfedge}() = \text{any of } e_0, e_9, e_{12}, e_{16}$
- (iv) $e_3 \rightarrow \text{face}() = f_4$
- (v) $e_{12} \rightarrow \text{vertex}() = V_0$
- (vi) $V_1 \rightarrow \text{halfedge}() = \text{either of } e_6, e_{10}$
- (vii) $e_{13} \rightarrow \text{next}() = e_3$
- (viii) $f_4 \rightarrow \text{halfedge}() = \text{any of } e_3, e_{12}, e_{13}$
- (ix) $e_0 \rightarrow \text{face}() = f_3$
- (x) $e_0 \rightarrow \text{next}() = e_{10}$
- (xi) $V_3 \rightarrow \text{halfedge}() = \text{either of } e_3, e_{14}$
- (xii) $e_3 \rightarrow \text{next}() = e_{12}$

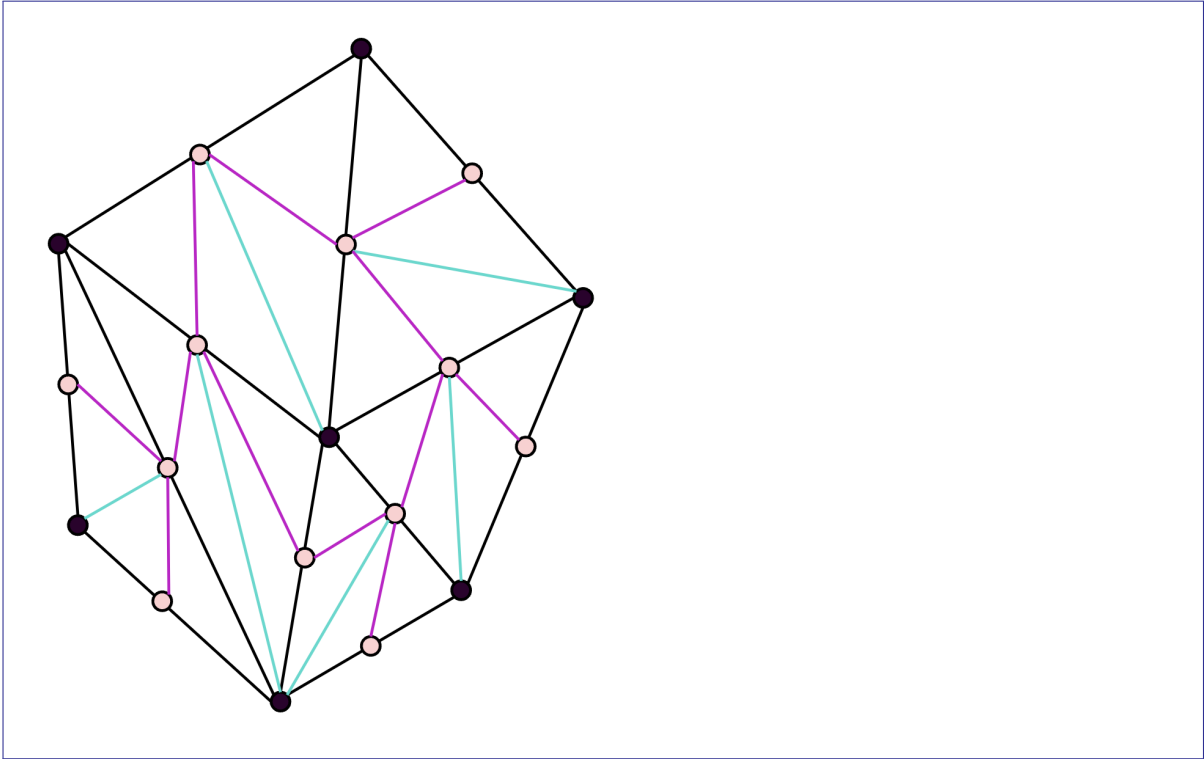
2 Loop-de-Loop

Loop subdivision is a method for upsampling triangular meshes. A single Loop subdivision involves:

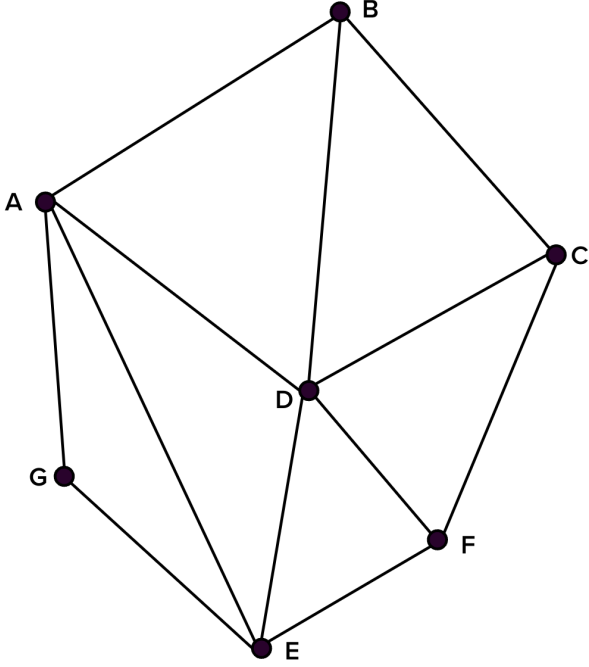
1. 4-1 subdividing each triangle (face splitting).
 2. Updating each vertex position (smoothing).
1. Draw the midpoint of each edge. Connect each edge's midpoint to the vertex (or vertices) opposite to that edge, effectively *splitting* on each edge.

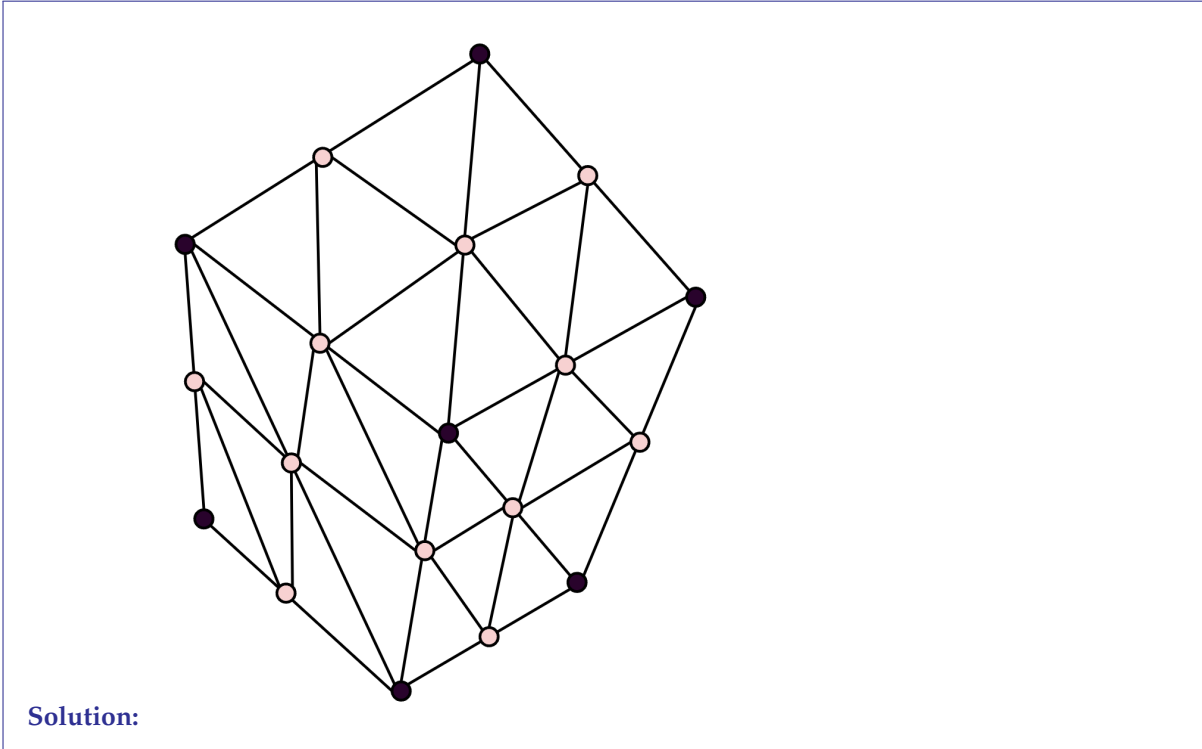


Solution: There are multiple right answers. The result depends on the order that the edges are processed in. Here is one example of a valid edge-splitting result:



2. For each new edge that you created in Part 1, if the edge connects a *new* vertex to an *old* vertex, perform an edge *flip*. Draw the updated mesh.





3. Now, let's smooth this new mesh using the Loop subdivision update rules.

Suppose $A = (2, 4, 0)$, $B = (5, 7, 0)$, $C = (7, 3, 0)$, and $D = (4, 2, 0)$. For the *new* vertex that is initialized as the midpoint of edge BD , calculate its updated position.

Solution:

$$\frac{3}{8}(B + D) + \frac{1}{8}(A + C) = \frac{3}{8}((5, 7, 0) + (4, 2, 0)) + \frac{1}{8}((2, 4, 0) + (7, 3, 0)) = \frac{3}{8}(9, 9, 0) + \frac{1}{8}(9, 7, 0) = \frac{1}{8}(36, 34) = (4.5, 4.25).$$

4. In general, when is the position of a *new* vertex unchanged by the Loop subdivision update rule? In other words, when is the updated position equal to the midpoint of the original edge?

Solution:

When $(B + D) = (A + C)$, so $\frac{3}{8}(B + D) + \frac{1}{8}(A + C) = \frac{4}{8}(B + D)$. In other words, when $\frac{1}{2}(B + D) = \frac{1}{2}(A + C)$, so the midpoint of BD is equal to the midpoint of AC , and $ABCD$ can represent the vertices of a parallelogram.

5. Vertex D is an *old* vertex that has neighbors A, B, C, E , and F . $E = (5, 0, 0)$ and $F = (6, 1, 0)$. Calculate the updated position of vertex D .

Solution: Here $n = 5$ and $\beta = \frac{3}{8n} = \frac{3}{40}$. Thus,

$$\begin{aligned} D' &= \left(1 - 5 \cdot \frac{3}{40}\right) \cdot (4, 2, 0) + \frac{3}{40} \cdot ((2, 4, 0) + (5, 5, 0) + (7, 3, 0) + (5, 0, 0) + (6, 1, 0)) \\ &= \frac{25}{40} \cdot (4, 2, 0) + \frac{3}{40} \cdot (25, 13, 0) \\ &= \frac{1}{40}(175, 89, 0) \\ &= (4.375, 2.225). \end{aligned}$$

6. In general, in what direction does the Loop subdivision update rule move an *old* vertex?

Solution: In the direction of the average of its neighboring vertices. This has a smoothing effect on the mesh.

3 Ray-Surface Intersection

When rendering a 3D object, we must determine which parts of it are visible, where shadows fall, and how lighting interacts within the scene. A straightforward yet computationally expensive approach is to cast rays from the camera through each pixel, finding where those rays intersect the object's mesh. In general, a ray can intersect the mesh zero times, once, multiple times, or—if it lies exactly in a triangle's plane—infinately many times.

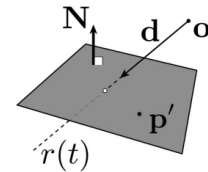
Recall that a ray is defined by its origin \mathbf{o} and its direction vector \mathbf{d} , and is parameterized by $t \geq 0$:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}.$$

1. As a warm-up, let's re-derive the equation for a ray intersecting an arbitrary plane. Recall that a plane can be defined as the set of all points \mathbf{p} satisfying

$$(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0,$$

where \mathbf{p}' is any point on the plane and \mathbf{N} is the plane's normal vector. Set \mathbf{p} equal to $\mathbf{r}(t)$ and solve for t .



Solution:

$$\begin{aligned}(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} &= 0 \\(\mathbf{o} + t\mathbf{d} - \mathbf{p}') \cdot \mathbf{N} &= 0 \\(\mathbf{o} - \mathbf{p}') \cdot \mathbf{N} + t\mathbf{d} \cdot \mathbf{N} &= 0 \\t\mathbf{d} \cdot \mathbf{N} &= (\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N} \\t &= \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}\end{aligned}$$

2. What does it mean if we get a value of $t < 0$?

Solution: This means that the intersection point with the plane is behind the ray's origin. Since the ray only moves forward in the direction of \mathbf{d} for positive values of t , $t < 0$ indicates that this value of t is not a valid intersection with the ray.

Note that this is true for any ray-surface intersection problem, not just with planes.

3. What does it mean if $\mathbf{d} \cdot \mathbf{N} = 0$?

Solution: When $\mathbf{d} \cdot \mathbf{N} = 0$, the ray's direction vector is perpendicular to the plane's normal vector \mathbf{N} . Therefore, the ray is parallel to the plane and will either intersect for all values of t or not intersect at all.

To see which is the case, substitute $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ into the plane equation $(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$:

$$(\mathbf{o} + t\mathbf{d} - \mathbf{p}') \cdot \mathbf{N} = 0 \implies (\mathbf{o} - \mathbf{p}') \cdot \mathbf{N} + t(\mathbf{d} \cdot \mathbf{N}) = 0.$$

But since $\mathbf{d} \cdot \mathbf{N} = 0$, we get

$$(\mathbf{o} - \mathbf{p}') \cdot \mathbf{N} = 0.$$

If $(\mathbf{o} - \mathbf{p}') \cdot \mathbf{N} = 0$, the entire line (extended ray) lies in the plane, yielding infinite intersections.
If $(\mathbf{o} - \mathbf{p}') \cdot \mathbf{N} \neq 0$, the ray never intersects the plane (zero intersections).

4. Given the following implicit representation of an ellipsoid and the definition of a ray, compute where (and at what parameter value(s) of t) the ray intersects the ellipsoid:

$$f(x, y, z) = \frac{(x - 2)^2}{4} + (y - 2)^2 + \frac{z^2}{4} - 1$$

$$\mathbf{r}(t) = (0, 0, 0) + t(1, 1, 0)$$

Start by substituting the ray $\mathbf{r}(t)$ into the function $f(x, y, z)$ to obtain $f(\mathbf{o} + t\mathbf{d})$.

Solution: Set $f(\mathbf{o} + t\mathbf{d}) = 0$ and solve for t . Then plug your value(s) of t back into the ray equation to find the corresponding point(s) of intersection. Finally, identify where the ray first hits the ellipsoid (i.e., the smallest positive t).

To get $f(\mathbf{o} + t\mathbf{d}) = 0$, we substitute $x = 0 + t \cdot 1 = t$, $y = 0 + t \cdot 1 = t$, and $z = 0 + t \cdot 0 = 0$. This gives us:

$$\frac{(x - 2)^2}{4} + (y - 2)^2 + \frac{z^2}{4} - 1 = 0$$

$$\frac{(t - 2)^2}{4} + (t - 2)^2 - 1 = 0$$

$$\frac{5}{4}(t - 2)^2 = 1$$

$$(t - 2)^2 = \frac{4}{5}$$

$$t = 2 \pm \sqrt{\frac{4}{5}}$$

Both t values are positive (since $\sqrt{\frac{4}{5}} < 2$), meaning they are both valid intersections. The first t value is the first intersection. Plugging in $t = 2 - \sqrt{\frac{4}{5}}$, we see that the ray first intersects the ellipsoid at $(0, 0, 0) + \left(2 - \sqrt{\frac{4}{5}}\right)(1, 1, 0) = \left(2 - \sqrt{\frac{4}{5}}, 2 - \sqrt{\frac{4}{5}}, 0\right)$.