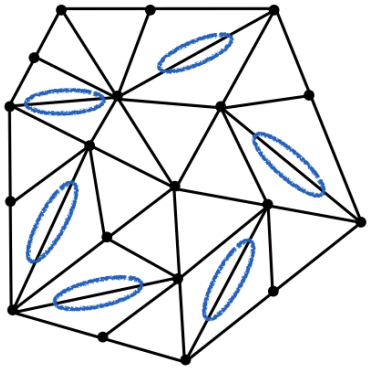
(1j) T ◯ F ◯ **[Soln: F/T]** If we perform an Edge Collapse mesh operation, we will delete two faces.
(This statement is only always true when working with triangle meshes, but is no longer the case with other shapes. However, because this class primarily works with triangle meshes in lecture, discussion, and projects, both answers receive credit)

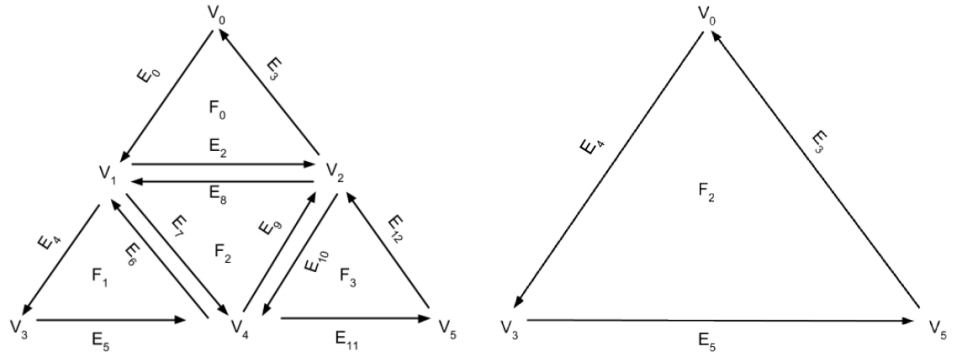
(1i) T ◯ F ◯ **[Soln: T]** If we start with N triangles, after K levels of Loop subdivision, we will have $N \cdot 4^K$ triangles.

(2l) (1 point) **T** When working with the half-edge data structure, Edge Flip, Edge Split and Edge Collapse operations can be implemented in $O(1)$ time.

Solution:



(4c) (4 points) Consider the following initial (left) and final (right) half-edge meshes.



For this problem, please use `halfedge()`, `vertex()`, `face()`, `twin()`, `next()` as the calls to both get and set those items. For example, given a halfedge pointer E_2, you could set its twin to the halfedge belonging to some vertex V_1 by writing:
$E_2 \rightarrow$ `twin()` $= V_1 \rightarrow$ `halfedge()`

Link, an aspiring 3D artist at Hyrule Inc., is trying to modify the mesh on the left to look like the one on the right. He is initially only given a pointer to the halfedge E_4. How can Link correctly assign E_4's new face, using the fewest number of calls? Assume he cannot access any elements that are not shown. Write this in one line.

Solution: $E_4 \rightarrow$ `face()` $= E_4 \rightarrow$ `next()` $\rightarrow$ `next()` $\rightarrow$ `twin()` $\rightarrow$ `face()`

(4d) (4 points) Again considering the above meshes, what is the fewest number of elements that must have their `next()` reassigned in order to *guarantee* that Link gets the correct mesh on the right? Assume there are no changes outside of the parts of the mesh shown.

Solution: 2, E_3 and E_5

$E_5 \rightarrow$ next() $= E_5 \rightarrow$ next() $\rightarrow$ twin() $\rightarrow$ next() $\rightarrow$ next() $\rightarrow$ twin() $\rightarrow$ next()
$E_3 \rightarrow$ next() $= E_3 \rightarrow$ next() $\rightarrow$ next() $\rightarrow$ twin() $\rightarrow$ next() $\rightarrow$ twin() $\rightarrow$ next()

☆
4d) only asks which next pointers need to reassigned.
other pointers also need to be re-assigned.
Here is a non-exhaustive list:

$E_3 \rightarrow$ face()
$E_3 \rightarrow$ vertex()
$V_5 \rightarrow$ halfedge()
$V_0 \rightarrow$ halfedge()
$E_4 \rightarrow$ vertex()
$E_4 \rightarrow$ face()

(2m) (1 point) **T** Computing the intersection of a ray and a sphere requires solving a quadratic equation.

(5a) [4 points] First, implement the simple, exhaustive approach. Write a function that finds all the `Photons` within the given ball. Update the `results` parameter with the photons found.

```
// This is a helper function you can call in your code.
// You do NOT need to implement this.
// This function returns the distance between two points.
float distance(const Point3D &a, const Point3D &b);

void FindPhotonsInBall(const vector<Photon> &photons,
                       const Ball &ball,
                       vector<Photon> &results)
{
    /* Your code answer goes here */
```

Solution:

```
void FindPhotonsInBall(const vector<Photon> &photons,
                       const Ball &ball,
                       vector<Photon> &results)
{
    for(Photon p : photons) {
        if (distance(p.pos, ball.center) < ball.radius) {
            results.push_back(p);
        }
```

```
    }
}
```