

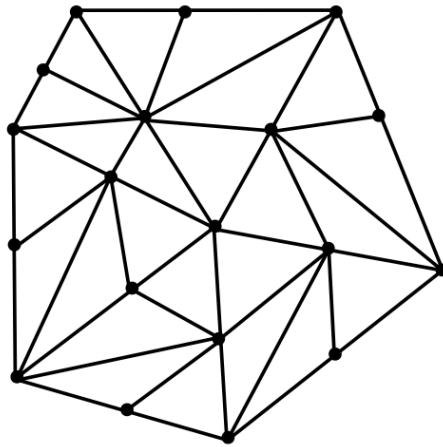
(1j) T F If we perform an Edge Collapse mesh operation, we will delete two faces.

(2g) (1 point) ___ In Loop subdivision starting with a mesh that has N triangles, after 4 levels of subdivision there are $16 \cdot N$ triangles.

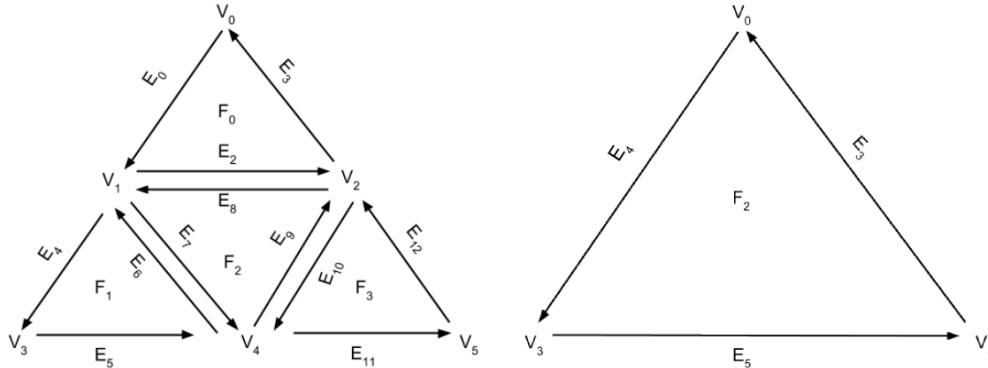
(2l) (1 point) ___ When working with the half-edge data structure, Edge Flip, Edge Split and Edge Collapse operations can be implemented in $O(1)$ time.

(3d) [6 points] Loop Subdivision

We studied a way to implement a round of Loop Subdivision on a triangle mesh. In the first step we split each edge, and in the second step we flip each new edge that connects an old point to a new point. Below you see such a mesh after the splitting step and before the flipping step. Circle all the edges that need to be flipped.



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



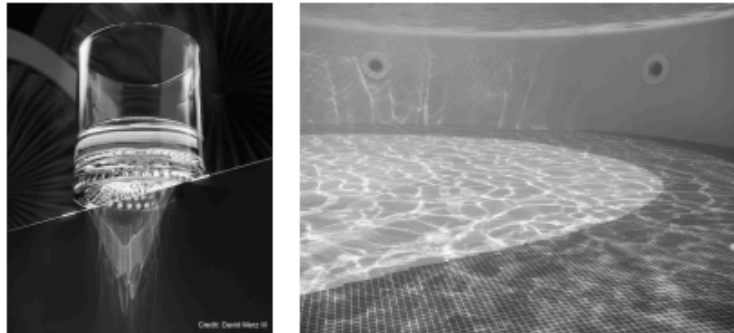
For this problem, please use `halfedge()`, `vertex()`, `face()`, `twin()`, `next()` as the calls to both get and set those items. For example, given a halfedge pointer E_2 , you could set its twin to the halfedge belonging to some vertex V_1 by writing:
 $E_2 \rightarrow \text{twin}() = V_1 \rightarrow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

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.

(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.

(1k) T F Computing the intersection between a ray and a sphere involves solving a quadratic equation.

5. (Total : 15 points) Geometric Acceleration Structures for Photon Mapping



Photon mapping is a global illumination algorithm invented by Henrik Wann Jensen. It excels at simulating caustics, which are spatial patterns of concentrated light caused by optical focusing of light, such as in the images above.

In photon mapping, millions of “photons” are traced from light sources through mirror / glass objects in the scene, refracting until they intersect diffuse surfaces, where the photon is stored at the intersection point along with metadata.¹ The photon map is used later in modified path-tracing algorithms, where the irradiance on diffuse surfaces is estimated by summing up photons near the ray-surface intersection point. This produces caustic patterns where photons are dense because of optical focusing.

A crucial geometric query in photon mapping is to find all the photons within a small ball – close to a given intersection point. In this problem, you’ll implement and accelerate this query function.

Here are the base structures you will use:

```
struct Point3D {
    float x[3];
};

struct Photon {
    Point3D pos;
    Metadata * data;
};

struct Ball {
    Point3D center;
    float radius;
};
```

(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 */
}
```

