Copilot

That’s a great question! Creating a separate `LinkedList` structure can indeed provide more flexibility and make your code more organized. This approach separates the concerns of managing the linked list (adding, deleting, traversing nodes) and the specifics of the data it contains (in this case, `Point` structs).

Here’s how you might define these structs:

```c
typedef struct Point {
    float x;
    float y;
} Point;

typedef struct Node {
    Point point;
    struct Node* next;
    struct Node* prev;
} Node;

typedef struct LinkedList {
    unsigned int size;
    Node* head;
    Node* tail;
} LinkedList;

typedef struct Polygon {
    LinkedList* vertices;
} Polygon;
```

In this setup, `LinkedList` is a general-purpose doubly linked list that can be used for other purposes in your program, not just for storing polygon vertices. The `Polygon` struct now contains a pointer to a `LinkedList` struct.

This approach does add a bit more complexity, but it can make your code more modular and easier to maintain. Plus, it’s a great exercise in structuring your code and working with pointers! 😊