# Existing Data Structure

point2D p;                  // coordinates of the vertex

  vertex\* prev;               // pointer to previous vertex

  vertex\* next;               // pointer to next vertex

  vertex\* neighbour;          // pointer to neighbouring vertex for intersection vertices

  bool source;                // to mark source vertices of the polygon

  bool intersection;          // to mark intersection vertices

  double alpha;               // to describe relative edge position of an intersection vertex

  IntersectionLabel label;    // type of intersection vertex

  EntryExitLabel enex;        // entry/exit "flag"

# Data Structure for GPU

4 arrays to represent input data (Polygon P and Polygon Q) defined using pointers and malloc.

double [sizeP] polyPX  
double [sizeP] polyPY

double [SizeQ] polyQX  
double [sizeP] polyQY

2 arrays for intersection points (polygon P intersection point and polygon Q intersection point) and alpha value defined using malloc

intersectionsP[i] represents [X,y,apha] distinct intersection points available at 3 times threadID

double [3\*sizeP\*sizeQ] intersectionsP  
double [3\*sizeP\*sizeQ] intersectionsQ

2 arrays to represent neighbors for intersectionsP and intersectionsQ

neighborP[i] represets [neighbor type]

neighbor types = {P1->0, Q1->1, I\_P->2, I\_Q->3}

int [sizeP\*sizeQ] neighborP  
int [sizeP\*sizeQ] neighborQ

if intersection is empty and neighbors exist, then P1=Q1 (V-intersection)

# Phase 1: Intersection in GPU

Each thread in GPU considers one edge from P, and Q, and perform

If intersection is true, intersection points are saved to intersections arrays with alpha values.

neighbors arrays save the neighbors for each intersection point.

## Phase 2: Intersection Point Labelling

label array to save labels for each intersection point (IntersectionLabel and EntryExitLabel)