BAB III

METODOLOGI PENELITIAN DAN PERANCANGAN APLIKASI

3.1 Metodologi

Langkah-langkah yang digunakan dalam metodologi penelitian terdiri dari,

1. Studi literatur

Mencari studi kasus yang serupa, kemudian mempelajari hasil yang didapat dari studi kasus tersebut, lalu dikembangkan menjadi solusinya, dan dibandingkan dengan kasus yang diteliti.

2. Perancangan

Rancangan penelitian dilakukan sebelum melakukan implementasi dan uji coba. Rancangan dilakukan untuk mencegah terjadinya kesalahan pada saat implementasi dan memudahkan pengembangan aplikasi. Perancangan pada aplikasi menggunakan diagram *flowchart.* Rancangan yang dibuat adalah rancangan triangulasi menggunakan algoritma *Flip*, *Sweep-Line*, dan *Sweep-Hull* untuk membuat *Delaunay Triangulasi 2D*.

3. Implementasi

Setelah memahami kasus melalui studi literatur dan merancang aplikasi untuk penelitian, tahap berikutnya adalah melakukan implementasi pada aplikasi. Tahap implementasi dilakukan dengan menerapkan algoritma *Flip*, *Sweep-Line*, dan *Sweep-Hull* dalam pembuatan *Delaunay Triangulation* 2D.

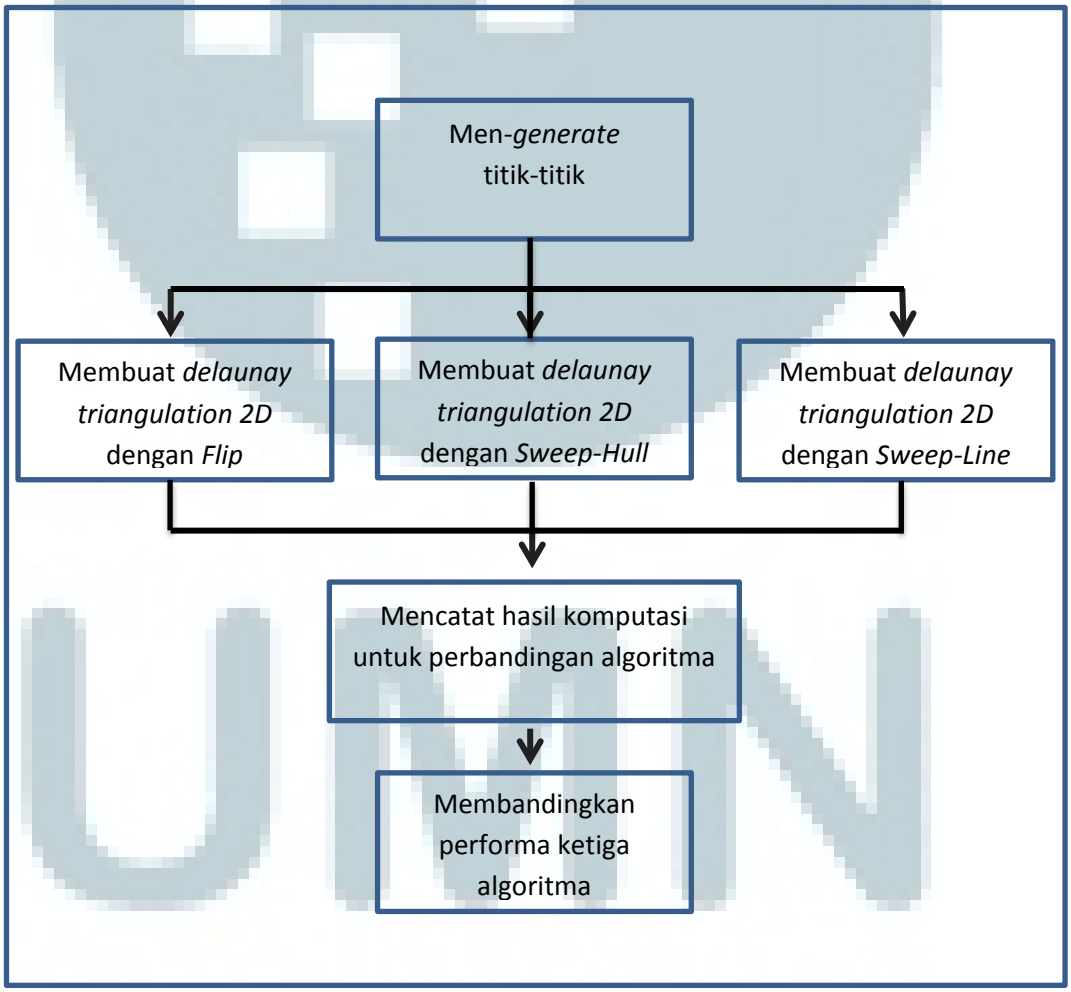
4. Uji Coba

Pengujian dilakukan terhadap hasil perbandingan algoritma *Flip*, *Sweep-Line*, dan *Sweep-Hull* dari segi *completeness*, *space complexity*, dan *time complexity*.

5. Menarik Kesimpulan

Setelah mendapatkan hasil perbandingan melalui uji coba, maka akan ditarik kesimpulan dari hasil perbandingan algoritma guna mendapatkan algoritma terbaik untuk pembuatan *Delaunay Triangulation* 2D.

Penelitian dilakukan dengan alur penelitian seperti terlihat pada gambar 3.1.



Gambar 3.1 Alur Penelitian.

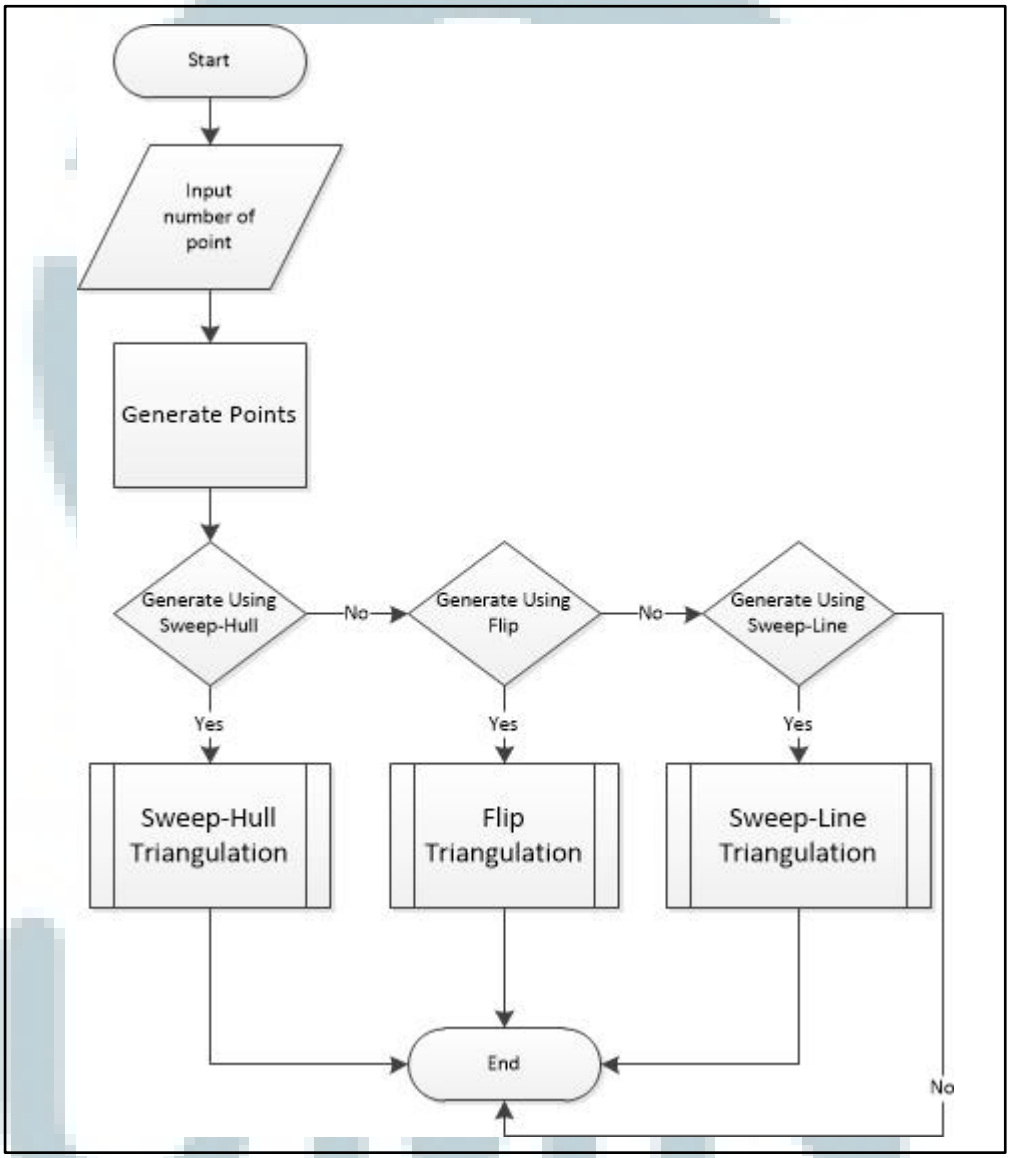
Pembuatan *Delaunay Triangulation* 2D diawali dengan men-*generate* titik-titik secara *unique*. Titik yang dihasilkan dengan *kordinat* x dan y mempresentasikan x sebagai garis *horizontal* dan y sebagai garis *vertical*. Dari titik-titik 2D yang dihasilkan dibuatlah *Delaunay Trianguation* menggunakan algoritma *Flip*, *Sweep-Line*, dan *Sweep-Hull*. Selanjutnya setelah proses pembuatan triangulasi selesai, dicatat hasil komputasi untuk kebutuhan perbandingan algoritma.

Perbandingan algoritma dilakukan dari segi *completeness*, *space complexity*, *dan time complexity* untuk menarik kesimpulan algoritma terbaik dalam pembuatan *Delaunay Triangulation* 2D.

3.2 Perancangan

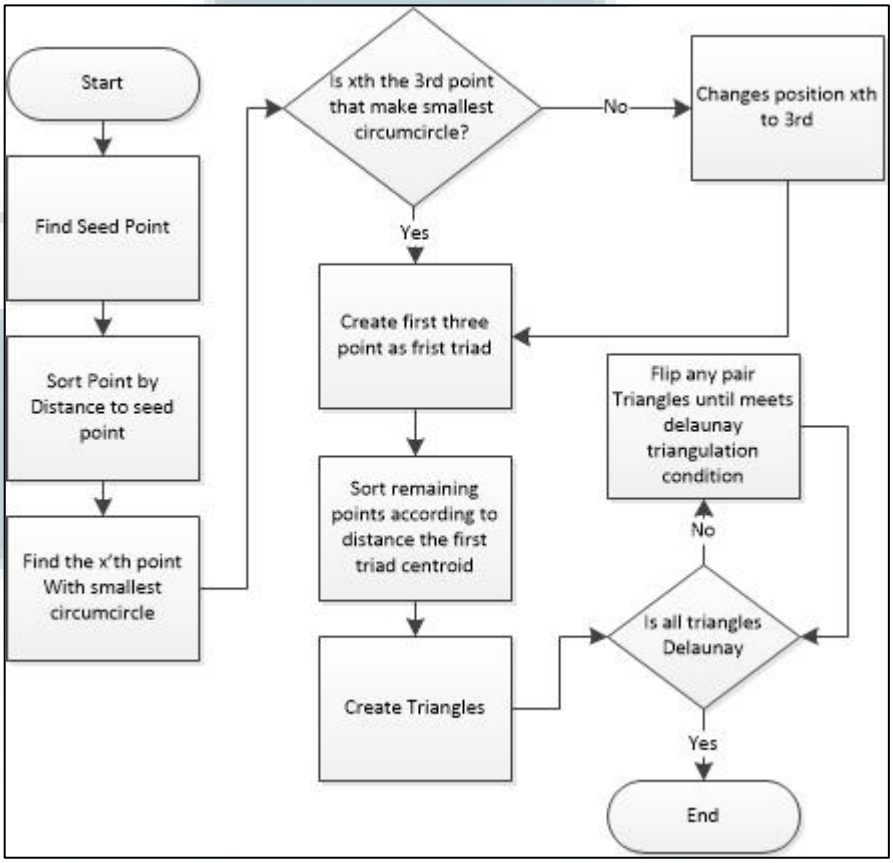
3.2.1 Flowchart

Perancangan aplikasi pembuatan *Delaunay Triangulation* 2D dilakukan dengan perancangan *conceptual flowchart* seperti yang terlihat pada gambar 3.2.



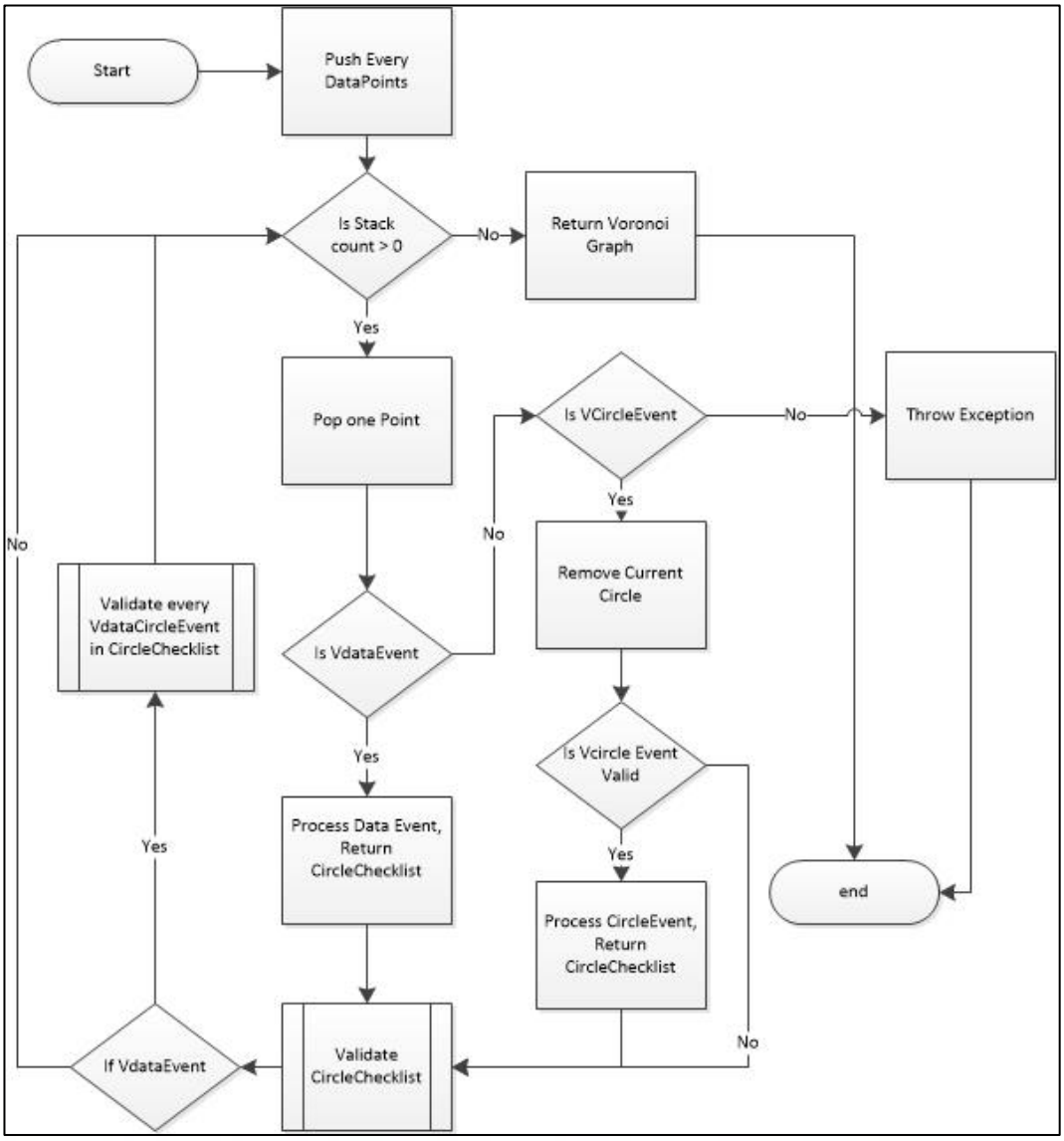
Gambar 3.2 *conceptual flowchart* aplikasi pembuatan *Delaunay Triangulation* 2D.

Gambar 3.2 menggambarkan aliran proses dari aplikasi *pembuatan Delaunay Triangulation* 2D proses *input* jumlah titik, *generate* posisi dari titik, dan memilih algoritma yang akan digunakan dalam pembuatan *Delaunay Triangulation* 2D.



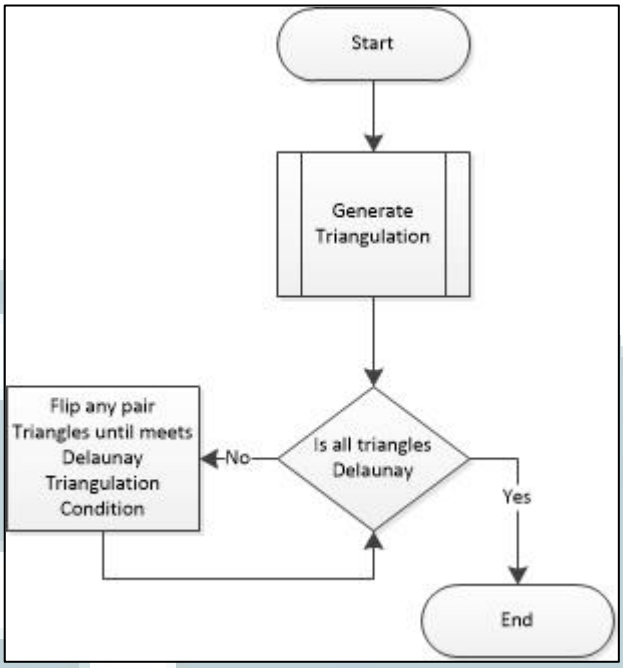
Gambar 3.3 *Flowchart Sweep-Hull triangulation*

Gambar 3.3 menunjukkan alur proses triangulasi menggunakan algoritma *Sweep-Hull*. Proses dimulai dengan menentukan *seed point* dari titik-titik yang ada dan membentuk triangulasi dengan menentukan dua titik berikutnya yang dapat membentuk *circumcircle* terkecil. Triangulasi pertama yang terbentuk dijadikan sebagai dasar untuk membentuk triangulasi dari titik yang tersisa. Implementasi *coding* dilakukan dengan menggunakan refrensi yang diperoleh dari *program* Phil Atkin's dalam s-hull.org.



Gambar 3.4 *Flowchart Sweep-Line triangulation*

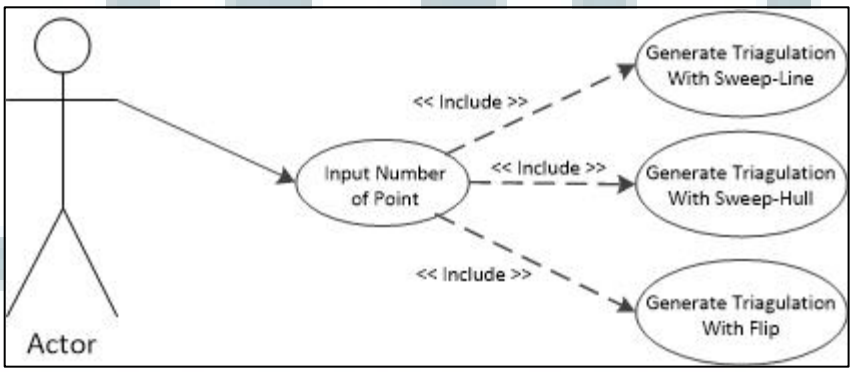
Gambar 3.4 menunjukkan alur proses triangulasi menggunakan algoritma *Sweep-Line*. Proses dimulai dengan memasukkan semua titik ke dalam *stack*. Setiap titik akan diproses untuk membentuk triangulasi. Proses berhenti ketika tidak ada lagi titik dalam *stack*. Dalam implementasi akan menggunakan *library* "FortuneVoronoi" yang meggunakan *Fortune's algorithm* untuk pembuatan triangulasi dan *voronoi diagram*.



Gambar 3.5 *Flowchart Flip triangulation*

Gambar 3.5 menunjukkan alur proses triangulasi menggunakan algoritma *Flip*. Proses dimulai dengan melakukan *generate triangulation*, hasil triangulasi akan diperiksa apakah sudah memenuhi syarat *Delaunay triangulation*. Jika sudah proses akan selesai, dan bila belum memenuhi syarat hasil triangulasi akan di-*flip* sampai semua segitiga memenuhi syarat *Delaunay Triangulation*. Proses *generate triangulation* dilakukan dilakukan dengan menggunakan algoritma yang di kembangkan oleh Paul Bourke's dalam *library* “Triangulator”.

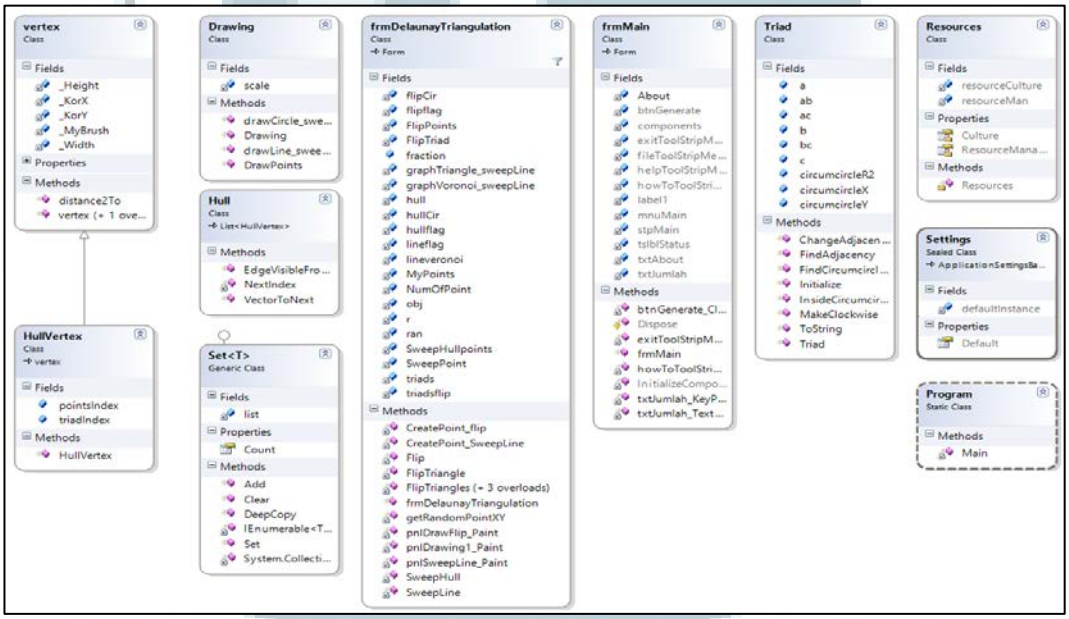
3.2.2 Use Case



Gambar 3.6 *Use Case*

Gambar 3.6 merupakan *use case* dari aplikasi yang akan dibuat, dimana seorang *Actor* dapat memberikan input angka, yang menunjukkan jumlah titik yang akan diproses. Setelah input diberikan *actor* baru bisa melakukan *Generate Trianguation* dengan menggunakan algoritma *Sweep-Hull*, *Sweep-Line*, dan *Flip*.

3.2.3 Class Diagram



Gambar 3.7 *Class Diagram*.

Gambar 3.7 merupakan gambar *class diagram* dari aplikasi yang dikembangkan. *Class vertex* adalah *class* yang menyimpan informasi titik-titik yang di *generate*. *Hull Vertex* merupakan *class* untuk titik *convex hull*. *Class triad* berfungsi untuk menyimpan informasi triangulasi. *Class Hull* dipergunakan untuk menyimpan informasi dari *hull vertex* untuk kebutuhan *maintain* titik dan *index triad*.

Set<T> merupakan *generic class* yang digunakan untuk kebutuhan pemrosesan data. *Class Drawing* digunakan untuk pembuatan *graphic* dalam aplikasi. frmDelaunayTriangulasi dan frmMain merupakan *class* dari *form* aplikasi.

Program, *Setting* dan *Resource* merupakan *class* hasil *inialisasi* bawaan dari aplikasi Visual Studio. Untuk kode implementasi *class* selain *class Vertex* akan di cantumkan di dalam lampiran.

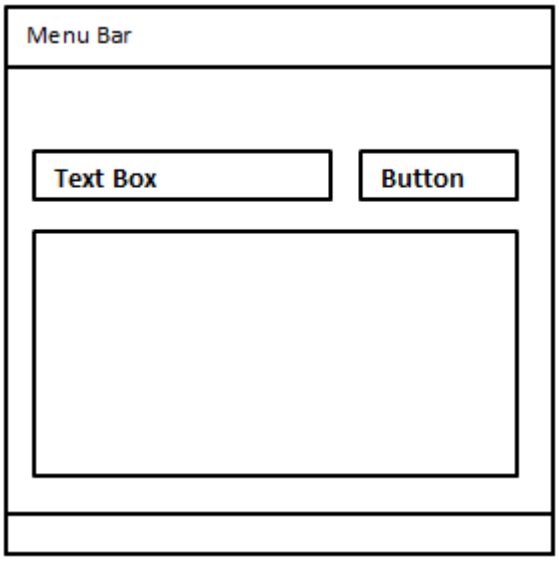
3.2.3.1 *Class Vertex*

Class Vertex digunakan untuk merepresentasikan setiap titik. *Class Vertex* memiliki lima parameter yaitu, _KorX, _KorY, _Width, _Height, dan _MyBrush. _KorX dan _KorY digunakan untuk menyimpan kordinat dari titik, dan _Width, _Height, dan _MyBrush digunakan untuk keperluan pembuatan *graphic* dari setiap titik. Selain itu setiap parameter memiliki sebuah *setter* dan *getter* untuk mengakses data, serta dua *constructor* untuk menginisialisasi tiap variable. Berikut ini adalah inialisasi class *Vertex*.

```
public class vertex{
        private float _KorX;
        private float _KorY;
        private int _Width;
        private int _Height;
        private Brush _MyBrush;

        #region setter and getter
        .
                .
        #endregion

        public vertex() {
            _Width = 4;
            _Height = 4;
            _MyBrush = new SolidBrush(Color.Red);
        }
        public vertex(float x, float y)
        {
            _Width = 4;
            _Height = 4;
            _MyBrush = new SolidBrush(Color.Red);
            this.x = x;
            this.y = y;
        }

        // Menghitung jarak antar titik
        public float distance2To(vertex other) {
            float dx = _KorX - other.x;
            float dy = _KorY - other.y;
            return dx * dx + dy * dy;
        }
    }
```

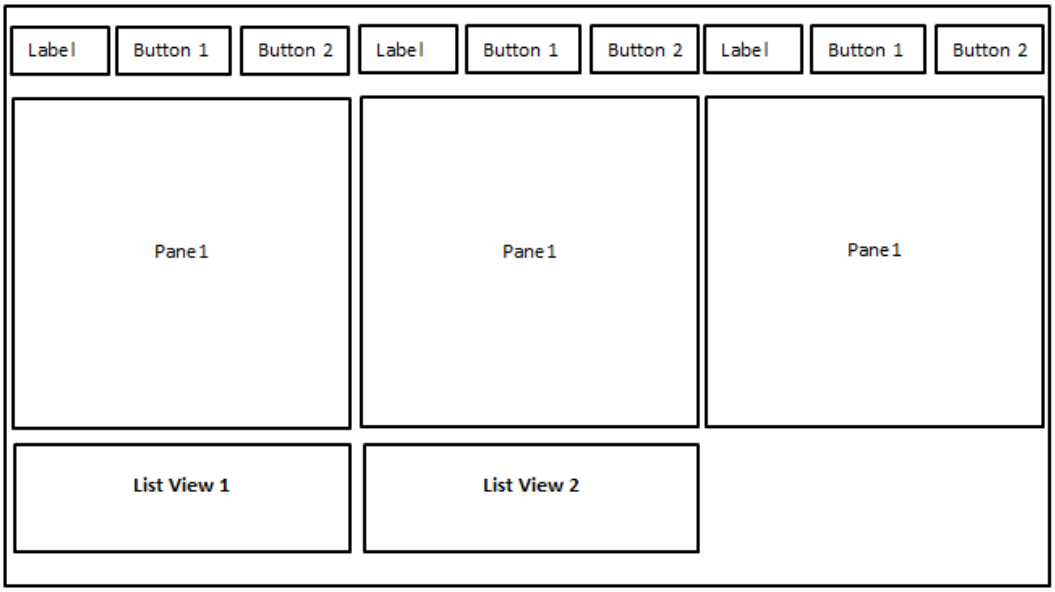
Gambar 3.8 *Class Vertex*

3.2.4 Sketsa Layar



Gambar 3.9 Sketsa halaman *input points*.

Gambar 3.9 merupakan sketsa halaman untuk *input points*, dimana *user* meng-*input* jumlah titik dari triangulasi yang akan dibuat. *Text Box* digunakan sebagai *input* titik. *Button* digunakan untuk mengirim jumlah titik dan menampilkan ke halaman *generate triangulation.*



Gambar 3.10 Sketsa halaman *generate triangulation*.

Gambar 3.10 merupakan halaman untuk *generate triangulation*. Label 1 gunakan untuk menampilkan nama algoritma yang digunakan untuk membuat triangulasi. *Button* 1 digunakan untuk memulai proses triangulasi dan hasil triangulasi akan ditampilkan di *panel* 1, hasil komputasi waktu dan jumlah *memory* yang digunakan akan ditampilkan pada *listview* 1 dan *listview* 2.

3.3 Perbandingan Algoritma

Ketiga algoritma akan dibandingkan berdasarkan tiga kriteria, yakni *completeness*, *space complexity*, dan *time complexity*. Perbandingan didapatkan melalui percobaan yang dilakukan beberapa kali untuk setiap algoritma dengan jumlah titik yang berbeda.

Untuk *completeness*, percobaan dilakukan dengan melihat dan mengamati secara manual untuk menentukan apakah solusi dapat ditemukan. Dalam proses generalisasi triangulasi ada kondisi yang menyebabkan solusi tidak sesuai dengan kondisi *Delaunay Triangulation*. Kondisi tersebut adalah ketika adanya jarak *circum circle* yang sama sehingga mempengaruhi proses pengurutan titik. Hal ini dapat diatasi dengan men-*generate* ulang triangulasi sampai memenuhi kondisi *Delaunay Triangulation*. Percobaan untuk membandingkan *completeness* akan dilakukan pada jumlah titik 10, 100, 250, 500, dan 1000. Percobaan akan diulang sampai memperoleh triangulasi yang memenuhi kondisi *Delaunay Triangulation*.

Untuk *space complexity*, perbandingan akan dilakukan dengan jumlah batas titik 1000. Percobaan akan dimulai dengan jumlah titik 10 untuk ketiga algoritma sampai batas maksimum yang telah ditentukan. Perbandingan akan dilakukan dengan perhitungan *space usage* dari ketiga algoritma.

Perbandingan *time complexity* akan dilakukan dengan menghitung berapa lama waktu yang dibutuhkan untuk menjalankan fungsi ketiga algoritma. Perhitungan waktu akan dimulai saat tombol ditekan dan berhenti saat proses menentukan triangulasi selesai. Percobaan akan menggunakan jumlah titik 10, 100, 250, 500, dan 1000.

Semua hasil percobaan akan dicatat dan digunakan untuk membandingkan performa dari algoritma *Flip*, *Sweep-Line*, dan *Sweep-Hull* secara keseluruhan.