Volume 3, Issue 11, November 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Application of the Triangulation of Polygon

Dr. Manju Mandot*
Associate Professor
Department of Computer Science & IT
JRN, Rajasthan Vidyapeeth (D) University
Udaipur (Rajasthan), India

Nikhil Mehta
BE Student
Department of Computer Science
NIT, Suratkal-Karnataka
India

Abstract – The decomposition of polygon into triangles is called triangulation of polygon. Polygon triangulation is an essential problem in computational geometry. Working with a set of triangles is in most cases faster than working with entire polygon. Various algorithm have been developed for triangulation with each having different time complexity order of O (n log n). In this paper, we mention a few applications of the algorithm. The implementation is done in c language.

INTRODUCTION

A classic problem in computer graphics is to decompose a simple polygon into a collection of triangles whose vertices are only those of the simple polygon. By definition, a simple polygon is an ordered sequence of n points, $\mathbf{V_0}$ through $\mathbf{V_{n-1}}$. Consecutive vertices are connected by an edge h ($\mathbf{V_i}$, $\mathbf{V_{i+1}}$), $0 \leq i \leq n-2$ and an edge ($\mathbf{V_{n-1}}$,$\mathbf{V_0}$) connects the first and last points. Each vertex shares exactly two edges. The only places where edges are allowed to intersect are at the vertices. Different types of polygon are shown in Fig.1.

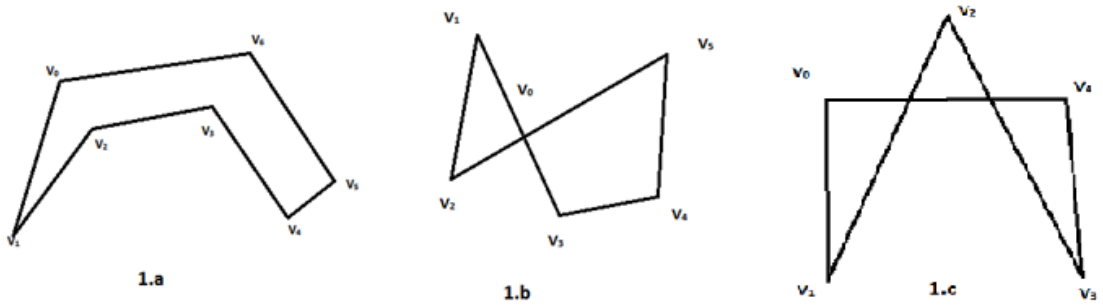
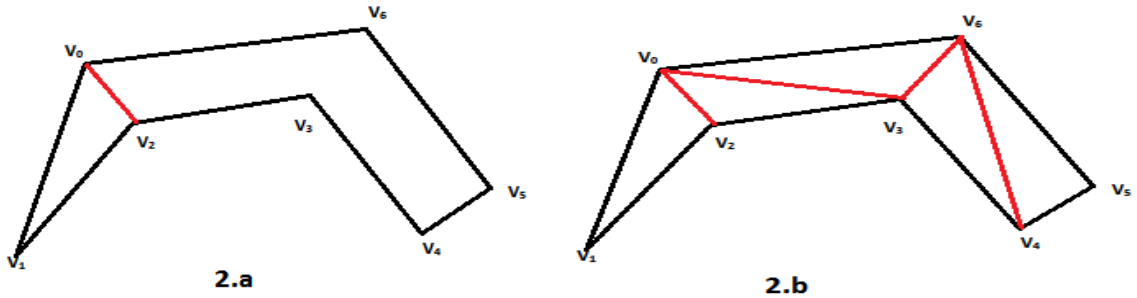


Fig. 1.a Represents simple polygon. Whereas 1.b & 1.c represents non simple polygons .
Fig 1.b Not simple polygon because it has a vertex which is common to more than three vertices
Fig. 1.c The edge is intersected by points which are not vertices.

If a polygon is simple, as we traverse the edges the interior bounded region is always to either side. Here in figure 1.a we assume that the polygon is counter clockwise ordered so that as you traverse the edges, the interior is to our left. The vertex indices in Figure 1.a for the simple polygon correspond to a counter clockwise order.

Diagonal Triangulation

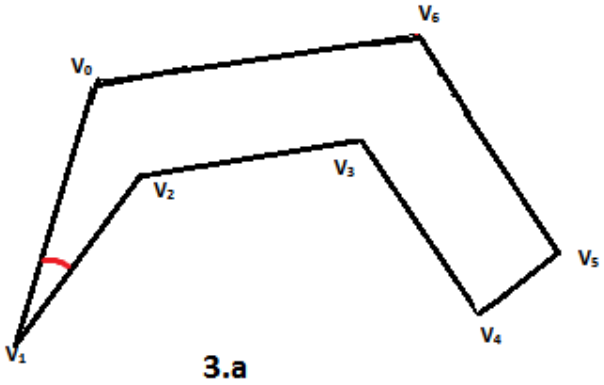
A line segment connecting two vertices of a polygon P that see each other in the way we say that no vertex or line segment would lie in between of the diagonal in the polygon P, such a line segment is called a **DIAGONAL.**



When we insert diagonals repeatedly (in the figure 2.a), polygon such that the diagonal do not intersect in the interior of the polygon, until we cannot insert more diagonals. Thus this kind of partition result into triangles and the process is called **TRINGULATION** of polygons (refer figure 2.b).

PROPERTIES OF POLYGON

1. A polygon must have at least one convex vertex.
   A vertex is **convex** if the interior angle of the vertices is less than 180^o .



3.a

2. Every polygon with more than three vertices has a diagonal.
3. Every polygon P of n vertices may be partitioned into triangles by the addition of diagonals.

PROPERTIES OF TRINGULATION

1. Every triangulation of n vertices uses n-3 diagonals and consists of n-triangles.
2. Sum of internal angle of a polygon of n vertices is (n-2)π.

ALGORITHMS FOR TRIANGULATION

The decomposition of polygon into triangles is called triangulation of polygon. Various algorithms have been developed for triangulation, with each having different time complexity as n (no. of vertices of polygon). The simplest algorithm is ear cutting algorithm. The time complexity or order of the ear cutting algorithm is O (n^2). Another algorithm like siedel which has time complexity order of O(n log n). An improvement using an increment randomised algorithm produces O(n $\log^* n$), where $\log^* n$ is the iterated logarithm function , this factor is almost constant for large n , thus we notice that for all practical purposes the randomised algorithm is linear time. An O(n) algorithm exists in theory , but it is quite complicated. Hence no implementation is available. Polygon triangulation is an essential problem in computational geometry. Working with a set of triangles is in most cases faster than working with the entire polygon, as the answers to queries often can be found by searching only locally. Many algorithms assume that the polygon is already triangulated, and several would not be possible without the triangulation. Given a triangulation, other partitions such as convex or star polygons can easily be found in linear time. Some algorithms that rely entirely upon partitioned polygons are character recognition [O'Rourke], shading [Fournier & Montuno] and shortest path [Guibas et al.].

EAR CLIPPING ALGORITHM

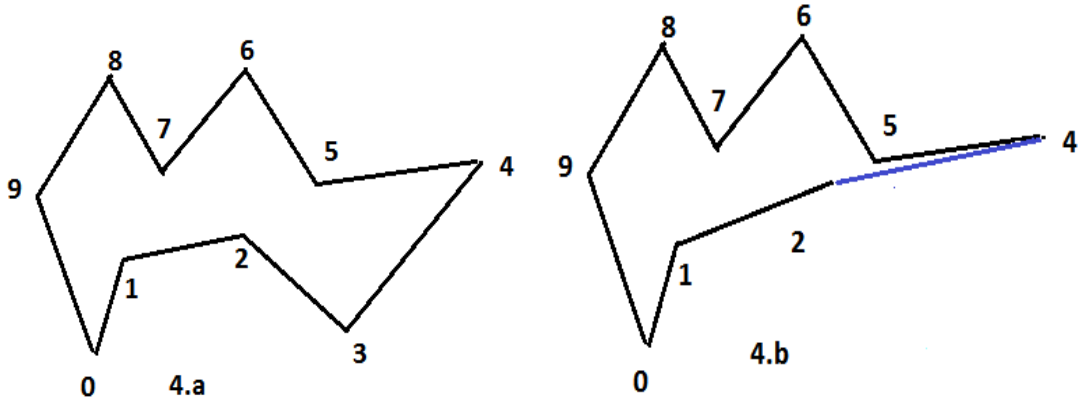
An **ear** of a polygon is the triangle formed by three consecutive vertices of a polygon, such that the vertices not common to the adjacent sides form a diagonal .And the diagonal does not intersect with the other edges or vertices of a triangle. One of the vertices is ear tip of the ear. A triangle consist of an ear tip, the ear tip can be at any of the vertices. A polygon with more than three sides has always at least two non overlapping ears. Thus this gives us a approach to the triangulation if we could locate an ear in polygon with n≥4 vertices and remove it, now the polygon left will be of n-1 vertices and thus we can repeatedly remove the triangles till we are left with the last triangle. If we implement this then we will get a n algorithm with order O (n^3). Now we if pay a little attention then we can reduce this algorithm to O (n^2) order. For this firstly we need to store the polygon vertices in a list so we can easily remove an ear tip whenever it is required. This process will take O (n) time. Secondly we iterate over the vertices and find the ears. For each vertex $\mathbf{V_i}$ and corresponding triangle $\mathbf{(V_{i-1},V_i,V_{i+1})}$ now all other vertices are tested whether they lie inside the triangle or not if they do not lie inside the triangle then we get an ear , here we consider 2 list reflex list and convex list . A **reflex vertex** is one for which the interior angle formed by the two edges sharing it is larger than 180 degrees. A **convex vertex** is one for which the interior angle is smaller than 180 degrees. We maintain four doubly linked lists for the polygon simultaneously. The vertices of the polygon are stored in a cyclical list, the convex vertices are stored in a linear list, the reflex vertices are stored in a linear list, and the ear tips are stored in a cyclical list.

Once the initial lists for reflex vertices and ears are constructed, the ears are removed one at a time. If V_i is an ear that is removed, then the edge configuration at the adjacent vertices V_{i-1} and V_{i+1} can change. If an adjacent vertex is convex, a quick sketch will convince you that it remains convex. If an adjacent vertex is an ear, it does not necessarily remain an ear after V_i is removed. If the adjacent vertex is reflex, it is possible that it becomes convex and, possibly, an ear. Thus, after the removal of V_i, if an adjacent vertex is convex you must test if it is an ear by iterating over and over the reflex vertices and testing for containment in the triangle of that vertex. There are O (n) ears. Each update of an adjacent vertex involves an earn test, a process that is O(n) per update. Thus, the total removal process is O (n^2).

Consider an example for Polygon Triangulation

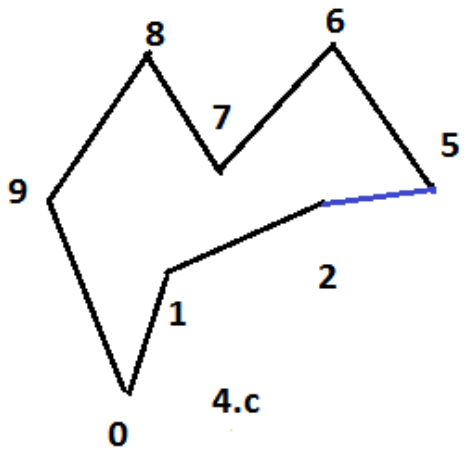
If we consider figure 4.a we get that we can form two list one of reflex angle and other of convex. Now the reflex list will contain ***R***={1,2,5,7} vertices and convex list will include ***C***={0,3,4,6,8}.Now we take vertices one by one and see if it is

convex we found that vertex 3 is convex and thus it is tested for ear and it is found to be a n ear . Thus the ear list will consist of ***E***={3,4,,6,8,9,0}. The vertex at 3 is removed, and thus we remove first triangle T_0 = {2,3,4}. Figure 4.b shows the reduced polygon with the new edge draw in blue.



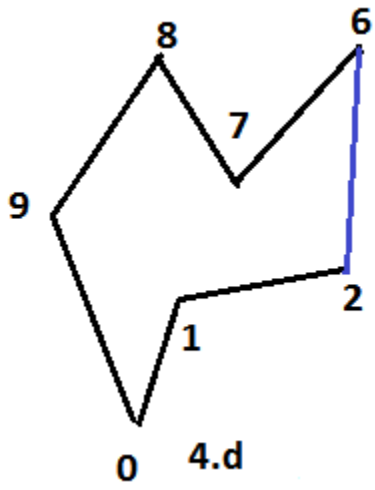
in Figure 4.b we can see that the ear {2,3,4} is removed from Figure 4.a and is replaced by a blue edge.

The vertex 2 was reflex and so it is now after removing the vertices. Thus vertex 4 which was an ear, Now ear list becomes ***E***={4,6,8,9,0}.
Now the ear at vertex 4 is removed and is replaced by an edge. Thus the triangle removed is T_1 ={2,4,5}.



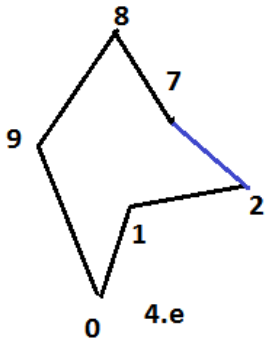
In figure 4.c shows a reduced polygon by removing T_1 = {2.4.5}

The vertex at 2 was reflex and still it is same. The vertex 5 was reflex but now it is convex. It is tested and found to be an ear. The vertex is removed from the reflex list ***R***={1,2,7} and is added to an ear ***E***={5,6,8,9,0}. Now the ear at vertex 5 is removed. We get new triangle T_2 = {2,5,6}.



In figure 4.d shows a reduced polygon by removing T_2 = {2.4.5}

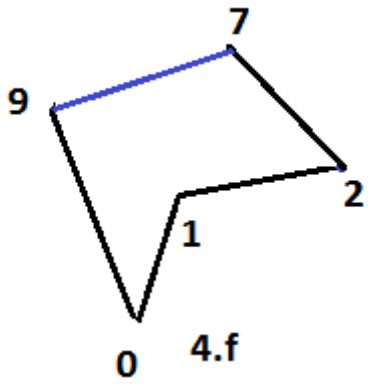
The vertex at 2 was reflex and now it is convex. It is difficult to tell from the figure but vertex 7 is inside triangle {1, 2, 6}. So vertex 2 is not an ear. Now we consider next ear vertex 6. Now the ear at vertex 6 is removed. We get new triangle T_3 = {2,6,7}.



4.e

In figure 4.e shows a reduced polygon by removing T_3 = {2.6.7}

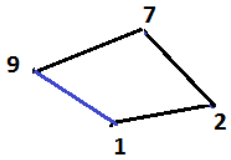
Now we see that the vertex 2 is forms an ear now, so it is removed from the convex list and is added to the ear list. The new ear list is now ***E*** = {8, 9, 0, 2}. Vertex 2 is so added because the list is circular list so it want make much difference we will get the triangulation in either way. Now we remove vertex 8 from the ear list and now .We get triangle T_4 = {7, 8,9}



4.f

In figure 4.f shows a reduced polygon by removing T_4 = {7.8.9}

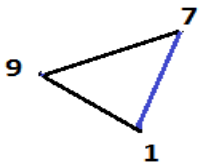
Now we see that vertex 7 forms an ear thus it is removed from the reflex list ***R***={1}, and is added to the ear list ***E***={9,0,2,1}. Now when we check for vertex 9 we find that it is not an ear , so it is removed from the ear list ***E***={0,2,7}. Now we remove vertex 0 from the polygon, and now we get another triangle T_5 = {9, 0, 1}.



4.g

In figure 4.g shows a reduced polygon by removing T_5 = {9, 0, 1}

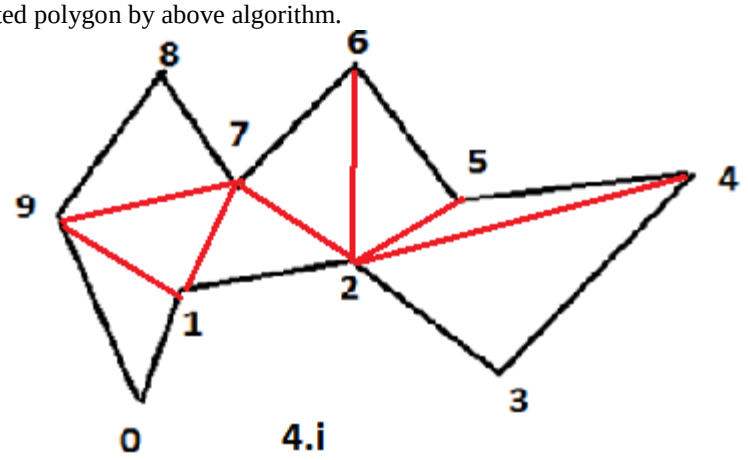
Now we are left with a simple structure now we remove the last triangle T_6={1,2,7}, and we get the final structure as given in the figure below.



4.h

In figure 4.h shows a reduced polygon by removing T_6 = {1, 2, 7}

Now we cannot triangulate furthermore, thus we get are polygon triangulated. By tracking each triangle back and joining them we get a triangulated polygon by above algorithm.



In figure 4.i shows a fully triangulated polygon.

Triangulations of Polygon with a Hole

The ear clipping algorithm may be applied to the polygon with holes. Such kind of a polygon will generally have two components i.e. inner and outer part. To Triangulate suck kind of polygon we should take into account that the vertices should be oppositely ordered in inner and outer polygons. For example if inner polygon is ordered in clockwise direction the outer must be ordered in counter clockwise direction.

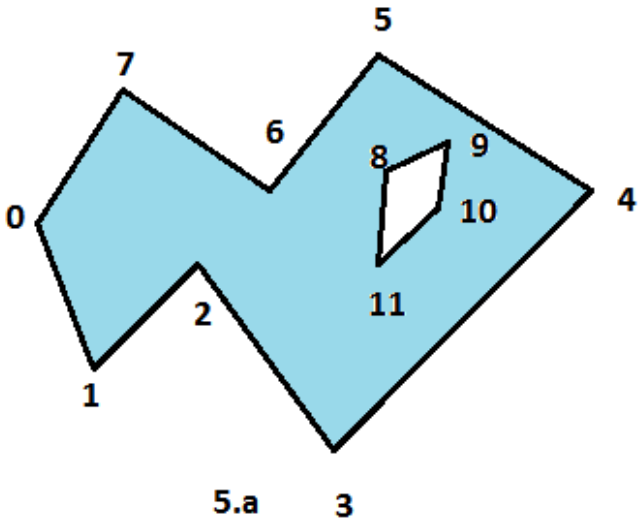


Figure 5.a shows a polygon with a hole

In figure 5.a we can see that the vertices are ordered in opposite order for inner and outer polygon. In above figure we see that vertex 10 and 4 are mutually visible. Now we introduce to edges from 10 to 4 (coinciding and opposite to each other) which will illustrate how triangulation will take place.

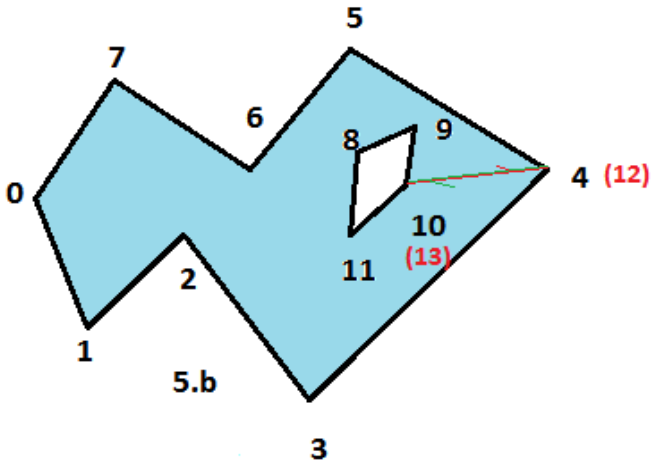


Figure5.b shows a polygon hole is removed by introducing two new edges that cut open the polygon Small arrows are attached to the edges to show their directions.

The mutually visible vertices must be duplicated in the sense that the vertex data structures are distinct. Each such data structure will store whether the vertex is convex or reflex. Even though a duplicated vertex has the same position as the original, one can be convex and the other reflex. In the vertex 4 and vertex 12, the original vertex was convex for the outer polygon. After duplication, vertex 4 associated with the green edge is a convex vertex but vertex 18 associated with the red edge is a reflex vertex.

The original outer polygon has vertices

{0, 1, 2, 3, 4, 5, 6, 7,}

and the original inner polygon has vertices

{8 , 9, 10, 11}

Vertex 4 is duplicated to produce vertex 12 and vertex 10 is duplicated to produce vertex 13. The polygon after introducing the two new edges is

{0, 1, 2, 3, 4, 10, 9, 8, 11, 13, 12, 5, 6, 7}

The new polygon may be triangulated by ear clipping.

SIEDEL ALGORITHM

In 1991 Seidel found a practical algorithm for triangulating simple polygons with an expected running time of $O(n \log^* n)$. . Several generalizations and optimizations of his routine are discussed, and the final result is an algorithm that can triangulate any set of overlapping and self- intersecting polygons and lines in the plane with near-linear expected running time. The implementation is completed with a set of functions that will graphically display any step of the algorithm.

Trapezoidation

When we extend a ray in horizontal in each direction from every vertex, and extend it till it hits another edge. Now we can easily see that we have divided the region into small trapezoid or triangle. It has an upper and a lower bound and a boundary on either side. If the length of upper and lower bound is zero then in this case the trapezoid will be a triangle. It is unique for each and every polygon and this process of trapezoiding the polygon is called tapezoidation.

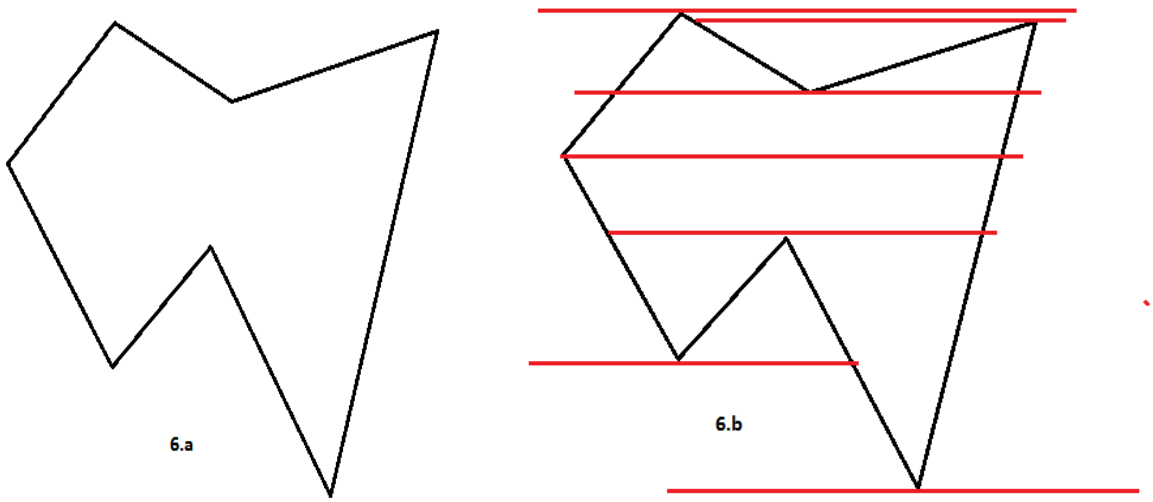


Figure6.a shows a polygon and figure 6.b shows a polygon with horizontal rays which are dividing the polygon into trapezoids

We can easily compute Triangulation in linear time of the given trapezoided polygon. We will triangulate only those trapezoids which are inside of the polygon, in the beginning the trapezoids are empty and do not contain any of the triangulated edge or even vertex. If the upper and lower boundary has same y coordinate or are lexicographic. Then we will consider the rightmost vertex as the lower one. No edge is treated as if it was horizontal, and no trapezoid will have more than one upper or lower bounding vertex. In some cases the two bounding vertices will be on the same side of the trapezoid, and in other they may be on opposite sides, or in the middle. Whenever the two vertices are not along the same edge, one can draw a diagonal between them without intersecting any edges since each trapezoid is empty. The addition of these diagonals is illustrated in figure 6.c. The resulting polygons bounded by edges and diagonals are monotone mountains.

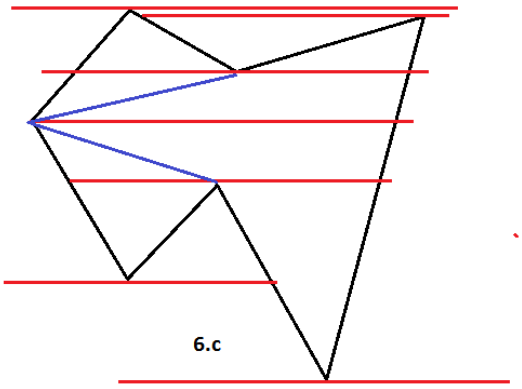


Figure 6.c shows a trapezoided polygon with diagonals.

Monotone Mountain has one edge called the base, which extends from the uppermost point to the lowermost point. The rest of the vertices form what is called a monotone chain, which is characterized by the fact that traversing the polygon from the top, every vertex will be lower than the previous one. If a monotone mountain is put with its base extending horizontally, it resembles a mountain range, hence its name.
Any convex vertex of the monotone chain is the tip of an ear, with the possible exception of the vertices of the base. Cutting such an ear off from a monotone mountain will always result in a smaller monotone mountain. This leads to a simple linear procedure for triangulating such polygons: Start with a list of all these convex vertices, and run through the list, cutting off each convex vertex and creating a triangle. For each vertex that is being cut, if its neighbours were not already convex, see if they have now become convex, and if either one has, add it to the end of the list. Once we have reached the end of the list, we are done. Needless to say, the implementation contains this algorithm.

Plane Sweep method:
The above method works in following ways.
1. Sort all the vertices either from left to right or top to bottom. For the example we have taken a polygon whose vertices are sorted from top to bottom.
2. The trapezoidal lines will be removed and we will traverse the divided polygon.
3. Now we will consider a horizontal sweep line which will be sweep from top till the bottom of the polygon.
4. If by adding a diagonal if we get a triangle inside den a diagonal will be added else it will be left as it is.
5. At the end of the polygon we will get the final triangulated polygon.

We can easily compute that by traversing in this we can get a polygon triangulated in O(n log n) time.

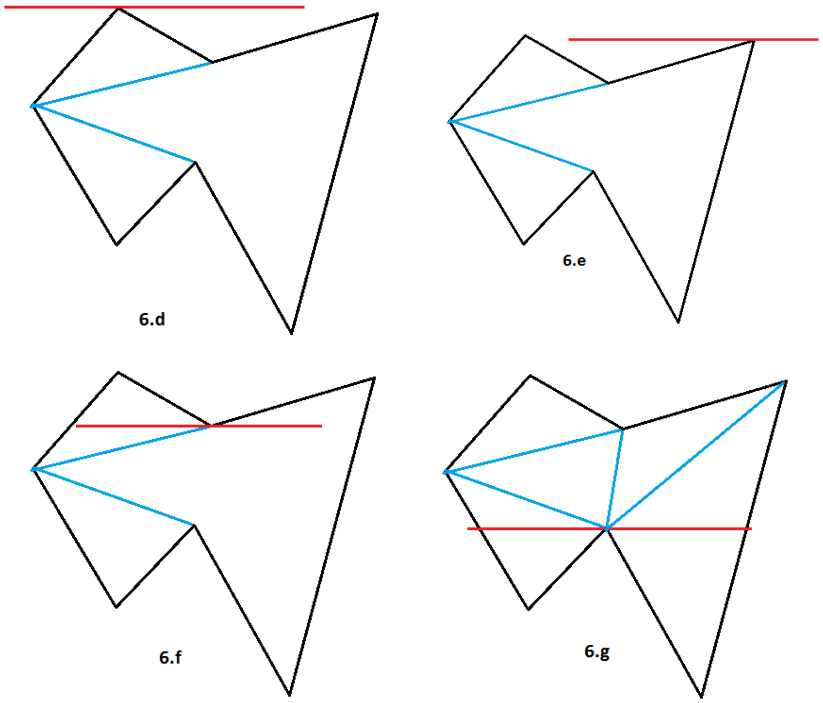


Figure 6.d shows a red sweep line. figure 6.e and 6.f shows how the line is swap checking for triangles and in figure 6.g we get the desired diagonals to complete the process

Application of TRIANGULATION

Famous Art gallery problem: it can be solved by triangulation. The problem goes like this: we are given with art gallery polygon of random shape and we need to know how many guards will be required to guard the art gallery. The restriction to this problem is that every guard is at a fix point and can guard 360°. Like as a example we take following room and determine how many guard will be required by this room.

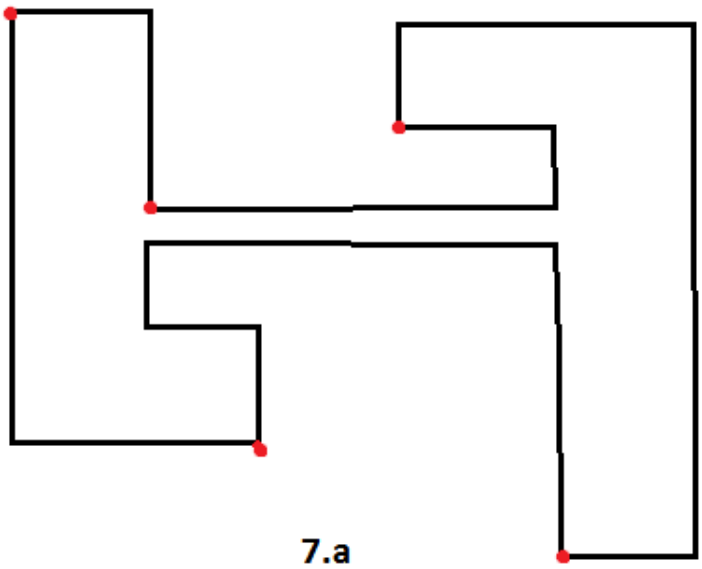
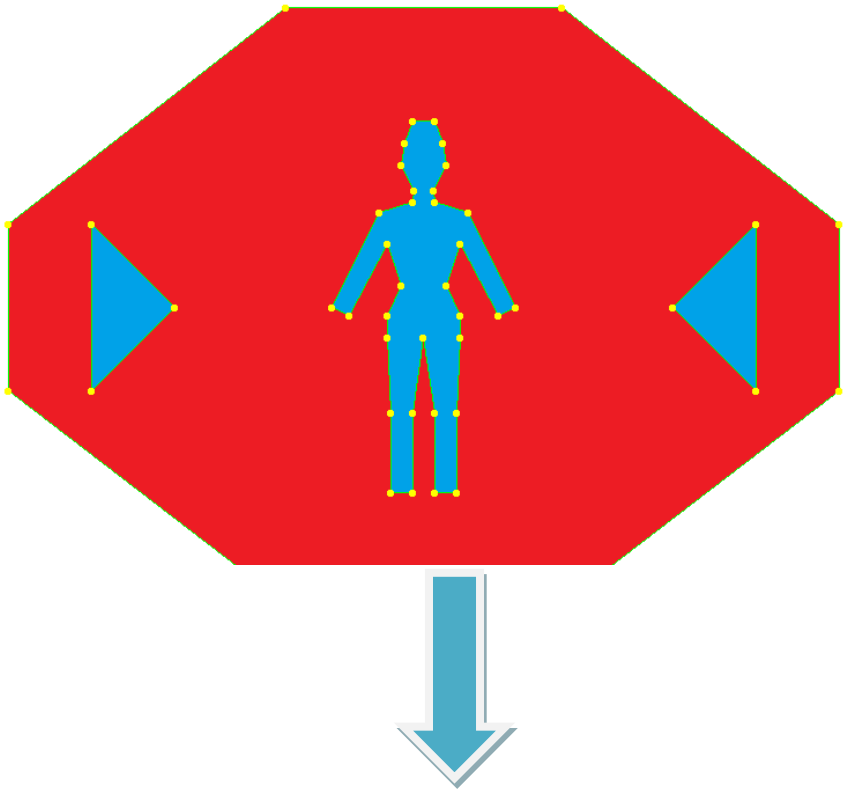


Figure 7.a shows a art gallery room with 5 guards.

We can see in above figure that it is not require keeping guards at each and every vertex. With little lesser number we can guard the whole art gallery. This is done via triangulation we triangulate the art gallery and see the vertex visibility. If the vertex is not visible then we place a guard on that vertex. The red dots in the above figure represent the guards. And thus the problem is solved.

Image Triangulation:

a) A multiply-connected polygonal area with three holes, and its triangulation. This can be thought as of an image. If we have an image and that can be divided in a set of polygon vertices then the triangulation formed by this vertex would be similar if same kind of picture arises. So by matching the set of triangles in such cases we will be able to find that the image appers same thus this concept of image triangulation can be use as for image detection. This could be a drastic change in the image detection concept.



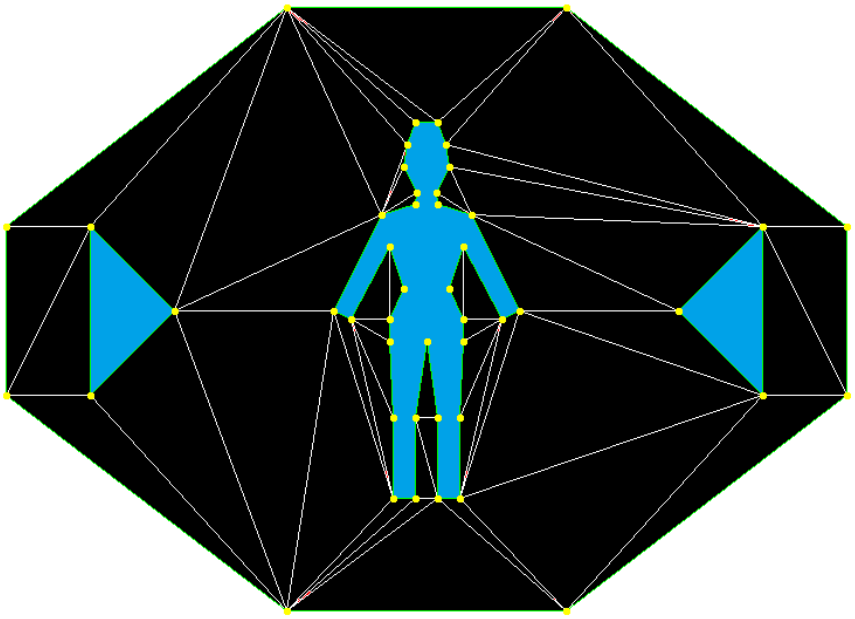
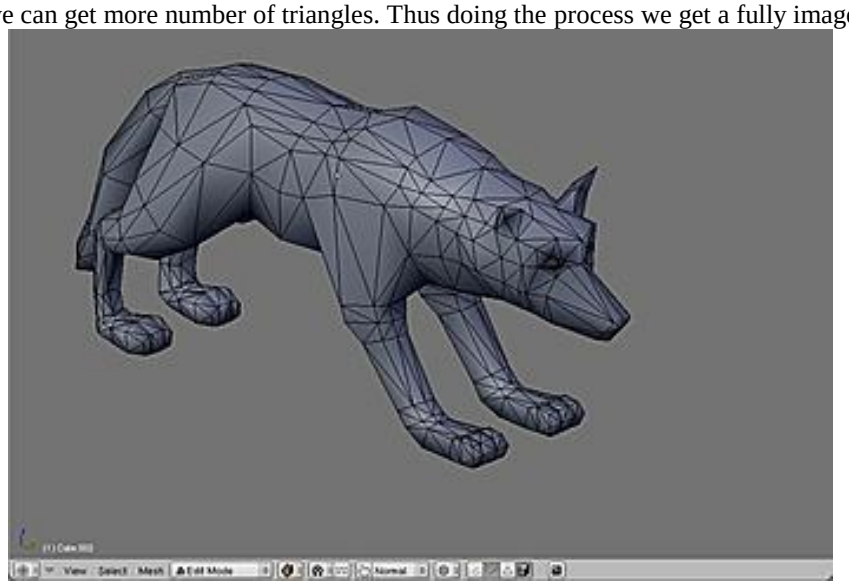


Figure 7.b shows an application part of the triangulation of polygon.

b) Creating a wire mesh of the image:

The concept of triangulation can also be used in preparing wire mesh of a product which leads to the basic structure of the object.

Wire mesh can be prepared by finding out the centroids of all the triangles and connecting them with the vertices of triangle thus we can get more number of triangles. Thus doing the process we get a fully image wire mesh.



Implementation of algorithm to triangulate a polygon

The implementation of this algorithm is done by using c with glut libraries. The implementation is done using various algorithms. The main aspect of the program is explained below

```
int main( int argc, char* argv[])
{
   int i;
   for(i=0;i<8;i++)
   di[i]=0;
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
   glutInitWindowPosition(0,0);
   glutInitWindowSize(1350,720);
   glutCreateWindow("nikhil");
   printf("%d",z);
   init();
   glutDisplayFunc(render);
```



```
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}
```

The above function is used to display a window which will be initial stage of our program once we have created a display screen we will focus on implementing the algorithm and try and find the triangulated polygon.
Now we will do following things in sequence.
First of all as our display screen is created we need to give it a basic design.
the below function will show how the basic structure is given to the screen.

```
void inwin()
{
    int i=0;
    glLineWidth(1);
    glColor3f(.8,.8,.8);
    glBegin(GL_POLYGON);
    glVertex2i(1150,0);
    glVertex2i(1350,0);
    glVertex2i(1350,720);
    glVertex2i(1150,720);
    glEnd();


    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1150,0);
    glVertex2i(1350,0);
    glVertex2i(1350,720);
    glVertex2i(1150,720);
    glEnd();


    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);

    glVertex2i(1200,320+di[5]);
    glVertex2i(1200,350+di[5]);
    glVertex2i(1315,350+di[5]);
    glVertex2i(1315,320+di[5]);
    di[5]=1;
    glEnd();


    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1200,20+di[0]);
    glVertex2i(1200,50+di[0]);
    glVertex2i(1315,50+di[0]);
    glVertex2i(1315,20+di[0]);
    di[0]=0;
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1200,80+di[1]);
    glVertex2i(1200,110+di[1]);
    glVertex2i(1315,110+di[1]);
    glVertex2i(1315,80+di[1]);
    di[1]=0;
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
```



```
    glVertex2i(1200,140+di[2]);
    glVertex2i(1200,170+di[2]);
    glVertex2i(1315,170+di[2]);
    glVertex2i(1315,140+di[2]);
    di[2]=0;
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1200,200+di[3]);
    glVertex2i(1200,230+di[3]);
    glVertex2i(1315,230+di[3]);
    glVertex2i(1315,200+di[3]);
    di[3]=0;
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1200,260+di[4]);
    glVertex2i(1200,290+di[4]);
    glVertex2i(1315,290+di[4]);
    glVertex2i(1315,260+di[4]);
    di[4]=0;
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1200,380+di[6]);
    glVertex2i(1200,410+di[6]);
    glVertex2i(1315,410+di[6]);
    glVertex2i(1315,380+di[6]);
    di[6]=0;
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(1200,440+di[7]);
    glVertex2i(1200,470+di[7]);
    glVertex2i(1315,470+di[7]);
    glVertex2i(1315,440+di[7]);
    di[7]=0;
    glEnd();
}
```

The above function would create the window of the as it looks below:

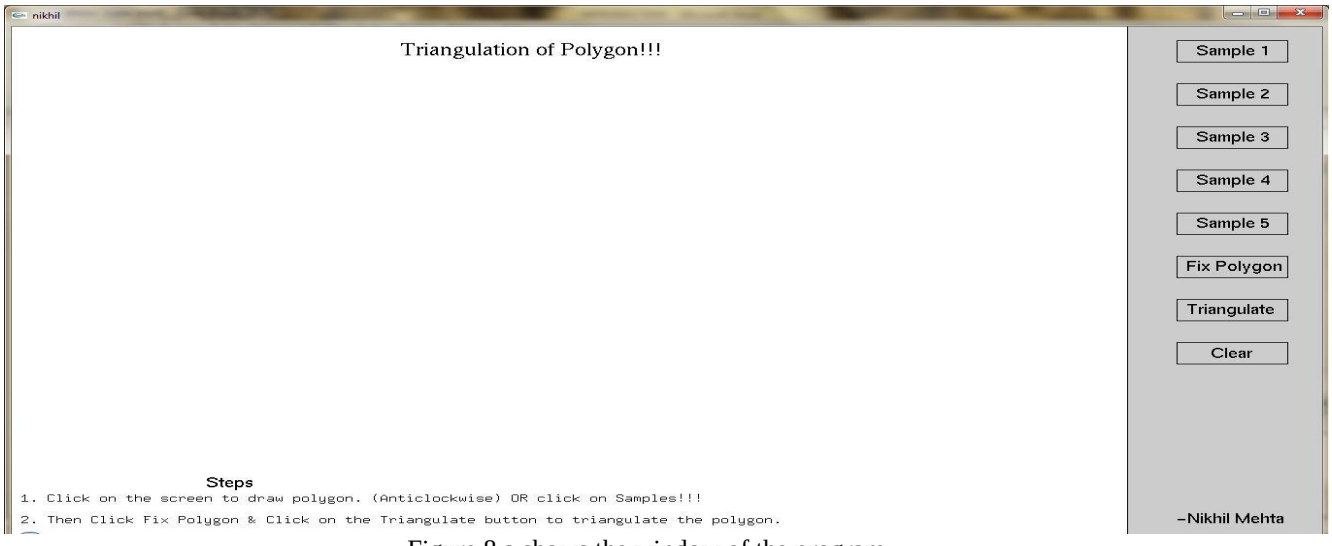


Figure 8.a shows the window of the program.

The text shown in the above widow is included in the function via drawtext command. Firstly we defined each text as shown below:

```
#define M_1 "Triangulate"
#define M_2 "Triangulation of Polygon!!!"
#define M_3 "1. Click on the screen to draw polygon. (Anticlockwise) OR click on Samples!!!"
#define M_4 "2. Then Click Fix Polygon & Click on the Triangulate button to triangulate the polygon."
#define M_5 "Steps"
#define M_6 "-Nikhil Mehta"
#define M_7 "Sample 1"
#define M_8 "Sample 2"
#define M_9 "Sample 3"
#define M_10 "Sample 4"
#define M_11 "Sample 5"
#define M_12 "Clear"
#define M_13 "Fix Polygon"
```

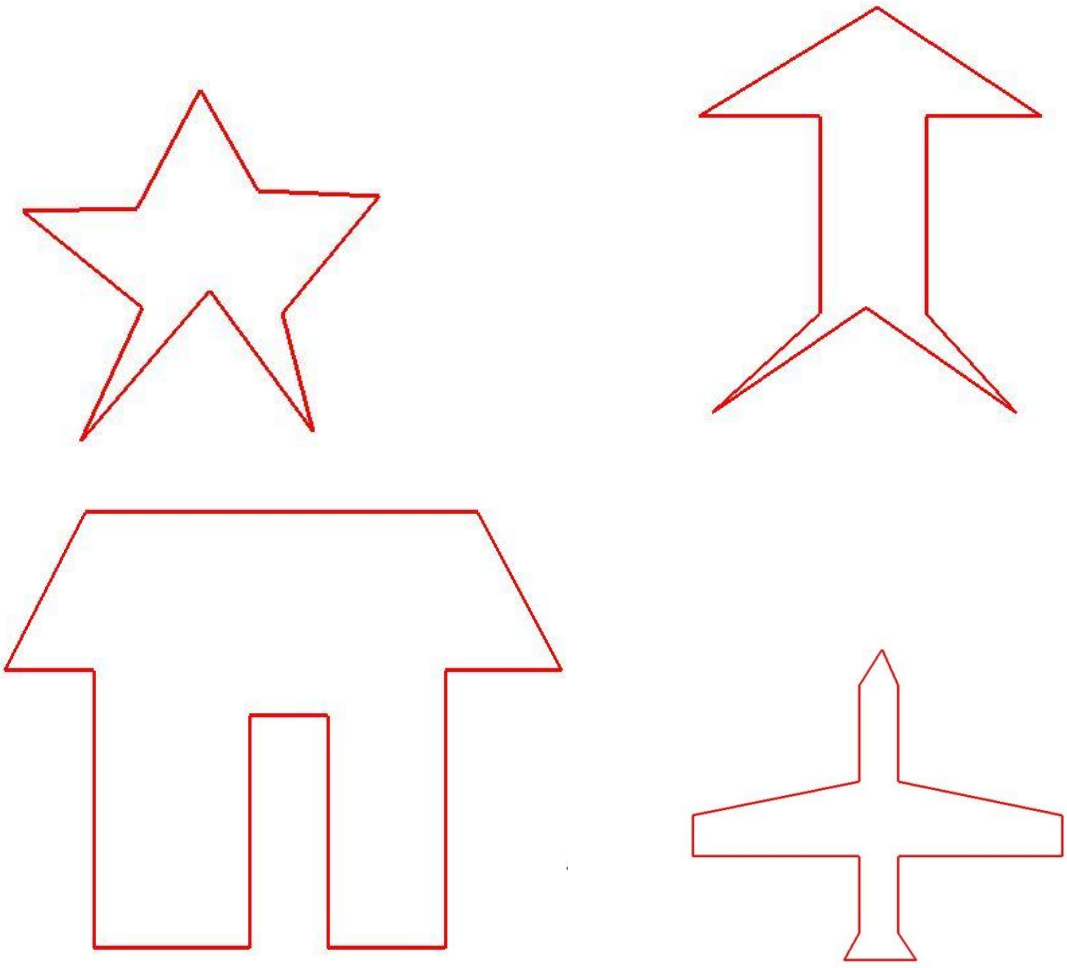
After including the text it we printed it in the displaying window using the below function:

```
void DrawText(const char *m_1,const char *m_2,const char *m_3,const char *m_4,const char *m_5,const char
*m_6,const char *m_7,const char *m_8,const char *m_9,const char *m_10,const char *m_11,const char *m_12,const
char *m_13)
{
        glColor3f(0,0,0);
        glRasterPos2i(1210,400);
  while(*m_1)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_1++);


        glColor3f(0,0,0);
        glRasterPos2i(400,40);
  while(*m_2)
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,*m_2++);

        glColor3f(0,0,0);
        glRasterPos2i(10,660);
  while(*m_3)
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15,*m_3++);

        glColor3f(0,0,0);
        glRasterPos2i(10,690);
  while(*m_4)
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15,*m_4++);

  glColor3f(0,0,0);
        glRasterPos2i(200,640);
  while(*m_5)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_5++);

    glColor3f(0,0,0);
        glRasterPos2i(1200,690);
  while(*m_6)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_6++);

        glColor3f(0,0,0);
        glRasterPos2i(1220,40);
  while(*m_7)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_7++);

        glColor3f(0,0,0);
        glRasterPos2i(1220,100);
  while(*m_8)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_8++);

        glColor3f(0,0,0);
```



```
        glRasterPos2i(1220,160);
  while(*m_9)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_9++);

        glColor3f(0,0,0);
        glRasterPos2i(1220,220);
  while(*m_10)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_10++);


  glColor3f(0,0,0);
        glRasterPos2i(1220,280);
  while(*m_11)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_11++);


        glColor3f(0,0,0);
        glRasterPos2i(1235,460);
  while(*m_12)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_12++);

  glColor3f(0,0,0);
        glRasterPos2i(1210,340);
  while(*m_13)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*m_13++);
}
```

Now the above window is created then we move on to taking input from the user so that we can get a polygon. The user inputs the polygon vertices by either clicking on window or choosing any of the five sample polygons.



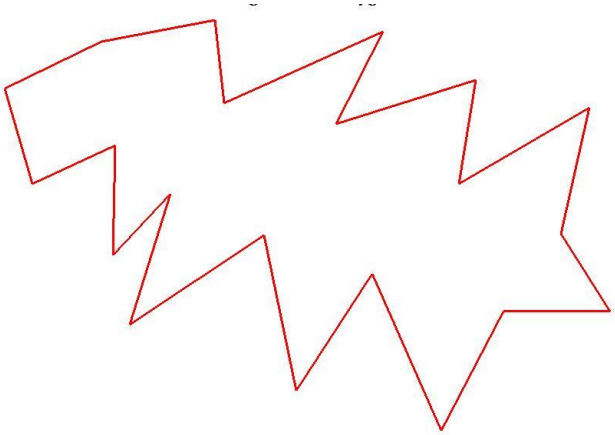


Figure 8.b shows the five sample polygons.

The user polygon can be formed by just clicking on window see below image for more reference.

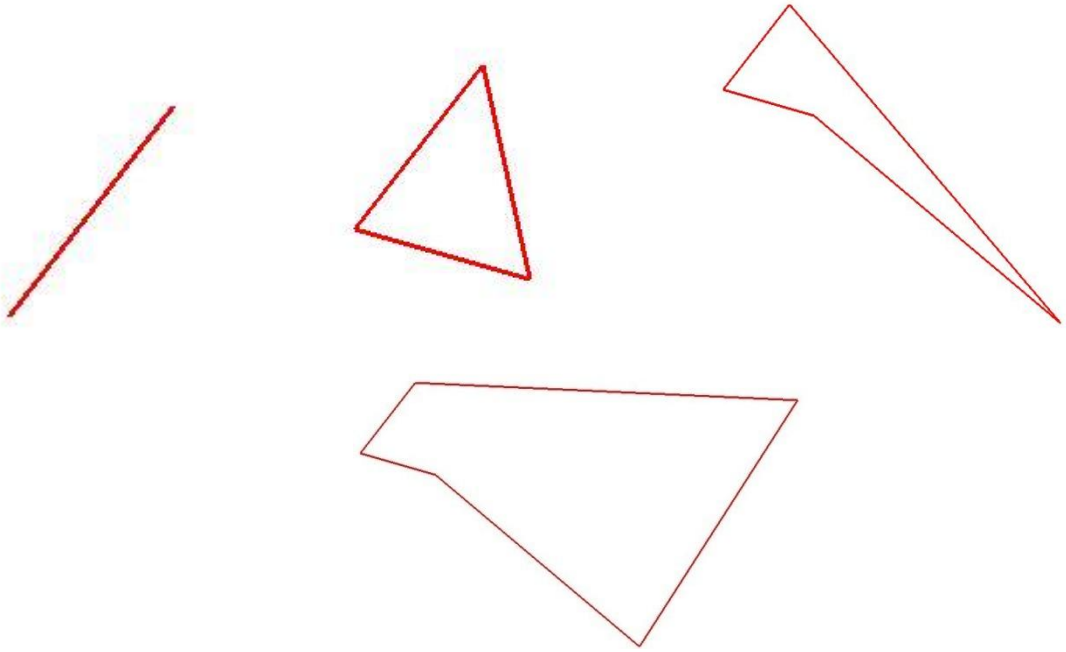


Figure 8.c shows how polygon are drawn by clicking on screen.

The code for forming sample polygon is give below:

```
void sample1()
{
   ax[0]=525;ay[0]=53+50;
   ax[1]=478;ay[1]=141+50;
   ax[2]=393;ay[2]=143+50;
   ax[3]=482;ay[3]=215+50;
   ax[4]=436;ay[4]=314+50;
   ax[5]=532;ay[5]=202+50;
   ax[6]=609;ay[6]=307+50;
   ax[7]=586;ay[7]=219+50;
   ax[8]=658;ay[8]=132+50;
   ax[9]=568;ay[9]=128+50;
   count=10;
}

void sample2()
{
   ax[0]=523;ay[0]=44+50;
   ax[1]=377;ay[1]=133+50;
   ax[2]=476;ay[2]=133+50;
```



```
   ax[3]=476;ay[3]=294+50;
   ax[4]=388;ay[4]=375+50;
   ax[5]=514;ay[5]=289+50;
   ax[6]=637;ay[6]=375+50;
   ax[7]=563;ay[7]=294+50;
   ax[8]=563;ay[8]=133+50;
   ax[9]=658;ay[9]=133+50;
   count=10;

}

void sample3()
{
   ax[0]=741;ay[0]=49+50;
   ax[1]=370;ay[1]=49+50;
   ax[2]=294;ay[2]=168+50;
   ax[3]=378;ay[3]=168+50;
   ax[4]=378;ay[4]=377+50;
   ax[5]=525;ay[5]=377+50;
   ax[6]=525;ay[6]=202+50;
   ax[7]=599;ay[7]=202+50;
   ax[8]=599;ay[8]=377+50;
   ax[9]=710;ay[9]=377+50;
   ax[10]=710;ay[10]=168+50;
   ax[11]=820;ay[11]=168+50;
   count=12;
}

void sample4()
{
   ax[0]=536;ay[0]=30+50;
   ax[1]=505;ay[1]=79+50;
   ax[2]=505;ay[2]=207+50;
   ax[3]=282;ay[3]=252+50;
   ax[4]=282;ay[4]=307+50;
   ax[5]=505;ay[5]=307+50;
   ax[6]=505;ay[6]=410+50;
   ax[7]=485;ay[7]=446+50;
   ax[8]=582;ay[8]=446+50;
   ax[9]=558;ay[9]=410+50;
   ax[10]=558;ay[10]=307+50;
   ax[11]=777;ay[11]=307+50;
   ax[12]=777;ay[12]=252+50;
   ax[13]=558;ay[13]=207+50;
   ax[14]=558;ay[14]=79+50;
   count=15;
}

void sample5()
{
   ax[0]=262;ay[0]=42+50;
   ax[1]=136;ay[1]=103+50;
   ax[2]=172;ay[2]=227+50;
   ax[3]=280;ay[3]=177+50;
   ax[4]=277;ay[4]=320+50;
   ax[5]=352;ay[5]=240+50;
   ax[6]=299;ay[6]=410+50;
   ax[7]=474;ay[7]=294+50;
   ax[8]=516;ay[8]=496+50;
   ax[9]=615;ay[9]=344+50;
   ax[10]=705;ay[10]=548+50;
   ax[11]=786;ay[11]=393+50;
   ax[12]=925;ay[12]=393+50;
```



```
   ax[13]=861;ay[13]=292+50;
   ax[14]=898;ay[14]=128+50;
   ax[15]=728;ay[15]=227+50;
   ax[16]=750;ay[16]=92+50;
   ax[17]=568;ay[17]=149+50;
   ax[18]=629;ay[18]=29+50;
   ax[19]=422;ay[19]=122+50;
   ax[20]=410;ay[20]=14+50;
   count=21;
}
```

To draw the polygon the below function is used.

```
void polygon()
{


        int i;

   glLineWidth(3);
   glColor3f(1,0,0);
   glPointSize(4);
   if (count==1)
   {
      glBegin(GL_POINTS);
      glVertex2i(ax[0],ay[0]);
      glEnd();
   }

   glBegin(GL_LINE_LOOP);
    for(i=0;i<count;i++)
    {

    printf("x=%d,y=%d\n",ax[i],ay[i]);
   glVertex2f(ax[i],ay[i]);
   }
   glEnd();

   z++;

}
```

Now once the polygon is created and is placed in an array we get all it its vertices . now we need to triangulate the polygon . for this first we need to find out and make a list of whether a vertex is a convex vertex or reflex vertex this done by finding the sine product of the vertex and adjacent vertices. thus we get an angle which reflects whether a particular vertex is convex or reflex. the function to determine this is given below:

```
void rc()
{
        int i,bx,by,cx,cy;
        ri=0,ci=0;
        float sqrb,sqrc,thetha,product;
        for(i=0;i<pi;i++)
        {
                if (poly[i]==poly[0])
                {
                        bx=ax[poly[pi-1]]-ax[poly[0]];
                        by=ay[poly[pi-1]]-ay[poly[0]];
      cx=ax[poly[1]]-ax[poly[0]];
                        cy=ay[poly[1]]-ay[poly[0]];
                }
                else if (poly[i]==(poly[pi-1]))
```



```
        {
          bx=ax[poly[i-1]]-ax[poly[i]];
          by=ay[poly[i-1]]-ay[poly[i]];
          cx=ax[poly[0]]-ax[poly[i]];
          cy=ay[poly[0]]-ay[poly[i]];
        }
      else
      {
        bx=ax[poly[i-1]]-ax[poly[i]];
                        by=ay[poly[i-1]]-ay[poly[i]];
        cx=ax[poly[i+1]]-ax[poly[i]];
                        cy=ay[poly[i+1]]-ay[poly[i]];
      }
   product=((bx*cy)-(by*cx));
          sqrb=sqrt((bx*bx)+(by*by));
          sqrc=sqrt((cx*cx)+(cy*cy));
          thetha=product/(sqrb*sqrc);
          printf("thetha=%f\n",thetha);
          if(thetha<0)
          {
            printf("%d",poly[i]);
   reflex[ri]=poly[i];
          ++ri;
          }
   else
          {
            convex[ci]=poly[i];
      ++ci;
          }
          }
          for(i=0;i<ci;i++)
          printf("convex =%d,ci=%d",convex[i],ci);
          for(i=0;i<ri;i++)
          printf("reflex =%d,ri=%d",reflex[i],ri);
}
```

Now after finding that the vertex is convex we check for each vertex if it is making triangle with its adjacent vertex or not if so a diagonal is added and a triangle is formed .

```
 void ear1()
{
   ei=0;
   int i, j,k=0,l;
 for (i=0;i<ci;i++)
 {
    l=convex[i];

   while(poly[k]!=l)
   k++;
   int cz=0;
   int pab,pac,pbc,abc;
   for(j=0;j<pi;j++)
   {
      if(poly[k]==poly[0])
      {
        if((poly[j]!=poly[k])&&(poly[j]!=poly[k+1])&&(poly[j]!=(poly[pi-1])))
        {
          abc=area(ax[poly[k]],ay[poly[k]],ax[poly[k+1]],ay[poly[k+1]],ax[poly[pi-1]],ay[poly[pi-1]]);
          pab=area(ax[poly[k]],ay[poly[k]],ax[poly[k+1]],ay[poly[k+1]],ax[poly[j]],ay[poly[j]]);
          pbc=area(ax[poly[j]],ay[poly[j]],ax[poly[k+1]],ay[poly[k+1]],ax[poly[pi-1]],ay[poly[pi-1]]);
          pac=area(ax[poly[k]],ay[poly[k]],ax[poly[j]],ay[poly[j]],ax[poly[pi-1]],ay[poly[pi-1]]);
```



```
        }
      }
      else if(poly[k]==(poly[pi-1]))
    {

      if((poly[j]!=poly[k])&&(poly[j]!=poly[k-1])&&(poly[j]!=poly[0]))
        {
          abc=area(ax[poly[k]],ay[poly[k]],ax[poly[k-1]],ay[poly[k-1]],ax[poly[0]],ay[poly[0]]);
          pab=area(ax[poly[k]],ay[poly[k]],ax[poly[k-1]],ay[poly[k-1]],ax[poly[j]],ay[poly[j]]);
          pbc=area(ax[poly[j]],ay[poly[j]],ax[poly[k-1]],ay[poly[k-1]],ax[poly[0]],ay[poly[0]]);
          pac=area(ax[poly[k]],ay[poly[k]],ax[poly[j]],ay[poly[j]],ax[poly[0]],ay[poly[0]]);
        }
    }
    else
    {
      if((poly[j]!=poly[k])&&(poly[j]!=poly[k+1])&&(poly[j]!=poly[k-1]))
        {
          abc=area(ax[poly[k]],ay[poly[k]],ax[poly[k+1]],ay[poly[k+1]],ax[poly[k-1]],ay[poly[k-1]]);
          pab=area(ax[poly[k]],ay[poly[k]],ax[poly[k+1]],ay[poly[k+1]],ax[poly[j]],ay[poly[j]]);
          pbc=area(ax[poly[j]],ay[poly[j]],ax[poly[k+1]],ay[poly[k+1]],ax[poly[k-1]],ay[poly[k-1]]);
          pac=area(ax[poly[k]],ay[poly[k]],ax[poly[j]],ay[poly[j]],ax[poly[k-1]],ay[poly[k-1]]);
        }
    }

    if((abc<(pab+pac+pbc+4))&&(abc>(pab+pac+pbc-4)))
    {
      cz=-1;

    }

    }
    if(cz==0)
    {
      ear[ei]=l;
      ++ei;
      printf("ear=%d\nei=%d\n",l,ei);
    }

 }


}

int area(int x1,int y1,int x2,int y2,int x3,int y3)
{
    float area;
    area=((x1*y2)+(x2*y3)+(x3*y1)-(x1*y3)-(x3*y2)-(x2*y1))/2;
    if(area<0)
    area=area*(-1.0);
    return area;
}
```

Now we have the list of vertices which form triangle with its adjacent vertices. Now we see that we need to update the triangle formed list and convex , reflex vertices so that we can triangulate the polygon completely so for this we use a function called update which will update the polygon and will triangulate accordingly.

```
void updatei(int i)
{
    int j,k;

    for(j=0;j<pi;j++)
    {
```



```
        if(poly[j]==i)
        {for(k=j;k<pi-1;k++)
        {
           poly[k]=poly[k+1];
        }
        break;

        }}
--pi;
        for(j=0;j<pi;j++)
        printf("poly[pi]=%d    ",poly[j]);

 for (j=0;j<pi;j++)
 {
    printf("ax=%d,ay=%d\n",ax[poly[j]],ay[poly[j]]);
 }


        rc();


    ear1();

    }
```

I have also used few more function to enhance the productivity of the code . Firstly I will show mouse function which is use for mouse interfacing.

```
void mouse(int button, int state,int mx,int my)
{

if ((button==GLUT_LEFT_BUTTON)&&(state==GLUT_UP))
{
   if((mx<1100)&&((point==0)||(point==1))&&(my>40)&&(my<600))
   {


   point =1;
   ax[count]=mx;
   ay[count]=my;
   printf("%d, %d, %d,%d ,%d\n",mx,my,z,point,count);
   ++count;
}
   else if((mx<1315)&&(mx>1200)&&(my>320)&&(my<350))

   {
     if(kj==0)
     {
        di[5]=1;
        point =2;

         kj++;
        }
   }
   else if((mx<1315)&&(mx>1200)&&(my>380)&&(my<410))
   {

   di[6]=1;

   if(f==1)
      steps2();
    else
      {
         point=4;
```



```
        kj++;
      }
   }
  else if((mx<1315)&&(mx>1200)&&(my>440)&&(my<470))

   {
      di[7]=1;
      point =3;
   }
      else if((mx<1315)&&(mx>1200)&&(my>20)&&(my<50))

   {
      clear();
      di[0]=1;
      sample1();
      point=2;
      polygon();
   }
      else if((mx<1315)&&(mx>1200)&&(my>80)&&(my<110))

   {
      clear();
      di[1]=1;
      point=2;
      sample2();
      polygon();
   }
      else if((mx<1315)&&(mx>1200)&&(my>140)&&(my<170))

   {
      clear();
      di[2]=1;
      point=2;
      sample3();
      polygon();
   }
      else if((mx<1315)&&(mx>1200)&&(my>200)&&(my<230))

   {
      clear();
      di[3]=1;
      point=2;
      sample4();
      polygon();
   }
      else if((mx<1315)&&(mx>1200)&&(my>260)&&(my<290))

   {
      clear();
      di[4]=1;
      point=2;
      sample5();
      polygon();
   }

   }

glutPostRedisplay();
}
```

Also i have used clear function to reset all the values of the code to initial stage.


```
void clear()
{
   glColor3f(1,1,1);
   glBegin(GL_POLYGON);
   glVertex2i(0,0);
   glVertex2i(1100,0);
   glVertex2i(1100,720);
   glVertex2i(0,720);
   glEnd();

   count=0;
   aj=1;kj=0,f=0;
   x=0;y=0;point=0;z=0;pi=0;ri=0;ci=0;ei=0;

}
```

To draw the diagonal between the triangle I have used diagonal function.

```
int diagonal()
{
   int i=0,a,b,c;
a=ear[0];

      if(a==poly[0])
      {

         b=poly[pi-1];
         c=poly[1];
   printf("\n1.\n");
      }
      else if(ear[0]==poly[pi-1])
          {

              b=poly[0];
              c=poly[pi-2];
   printf("\n2.\n");
          }
       else
       {
              for(i=0;i<pi;i++)
              if(a==poly[i])
              break;
          b=poly[i+1];
          c=poly[i-1];
       printf("\n3.\n");
       }

printf("\na=%d,b=%d,c=%d\n",a,b,c);
if(point==4)
{

        glColor3f(0,1,0);
   glBegin(GL_LINES);
   glVertex2i(ax[b],ay[b]);
   glVertex2i(ax[c],ay[c]);
   glEnd();
}
  //  printf("ei=%d",ei);
   return a;
}
```

Terminology and variables and Functions used in the program:

1. Libraries used in the code :
#include<stdlib.h> : for general use of the functions like printf(), scanf() etc.
#include<glut.h>: for graphics related functions like glvertex2i(), glbegin(), glend() , etc.
#include<math.h>: for mathematics related function like sqrt().

2.Global variables used:
x, y, point ,z ,pi ,ri ,ci ,ei ,ti ,kj ,aj ,f : This are either counters or used as flag in the code .
poly[100]: It is used to keep the track of no. of polygon in the program remained after triangulation.
ay[100],ax[100]: These two arrays are used to keep the record of the y and x coordinates of the vertices of the polygon.
reflex[100]: This array keeps the record of the reflex vertices in the polygon.
convex[100]: This array keeps the record of the convex vertices in the polygon.
count: It keeps the count of number of vertices in the given polygon.
ear[100]: This array keeps the record of the vertices which can be triangulated in the polygon.
di[8]: It is used for the particular boxes in the screen.

Functions used in the program:
main(): it is the main function which calls all the other functions.
render(): All the things which we project on screen are in this function.
inwin(): The initial window design in this function.
clear(): It is used to reset all values and clear the screen.
DrawText(): It is used for drawing text on screen.
rc(): It is used to find convex and reflex vertices.
ear1(): It is used for finding whether a vertex is forming traingle with its adjacent vertices.
area(): it is used to find areas of the triangle.
polygon(): it is used to draw the polygons.
diagonal():it is used to draw diagonals of the triangle.
updatei(): It is used to update the polygon in the code.
sample1(),sample2(),sample3(), sample4(), sample5(): All are used for storing the coordinates of the sample polygons.
mouse():this function is used for interfacing mouse with the program.

