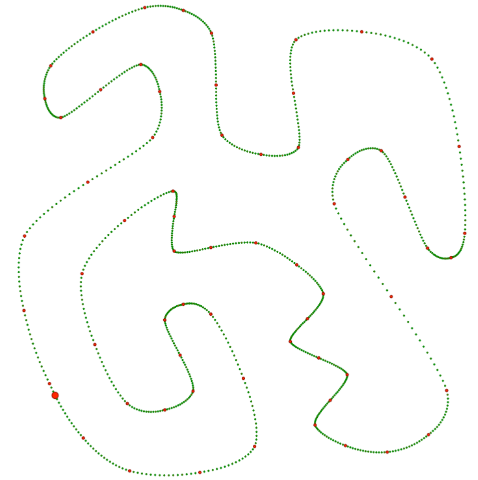
CS3451. Fall 08. Midterm. NAME: ___SOLUTION__ Grade:______ / 30.

Individual test. Sit sufficiently far from, other students and do not look at other students' work. Closed books. You can only have one page (your cheat-sheet) of notes. Please type and write legibly. Each question is 10 points. You should answer 3 out of any of the four questions. 24 points and above will correspond to an A at the midterm. You may answer the 4th question for extra credit.

1) **Animation**: You are given as input a planar polyloop P[] with vertex count n. Each vertex P[i] (small red dots in the figure) represents the position of a point at keyframe 16i. Provide the details of an implementation (Processing code) for animating the **cyclic motion** of a small disk (larger red dot) whose **smooth** trajectory has 16n frames (shown as small red and green dots) and **interpolates** the n original keyframes, thus using 15 intermediate frames (green dots) between each pair of consecutive red keyframes. Formulate your solution in terms of two procedure: **myInit**(), which will be executed once at initialization, and **myAnimate**(), which will be executed for each frame, as shown below. The code for n() and p(), which access the next and previous indices in P is provided. The code for linear interpolations S(A,t,B) is also provided. To set a point A to be equal to a point B, use A.setTo(B). P(A) returns a new copy of point A. If you call any other procedure please provide the Processing code for it. In **myInit**(), you may change the value of n and the content of P[]. You do not need to render the small red and green dots. MyAnimate must render the larger red dot for the current frame f so that the red dot keeps traveling along the smooth loop. Provide comments to facilitate reading.



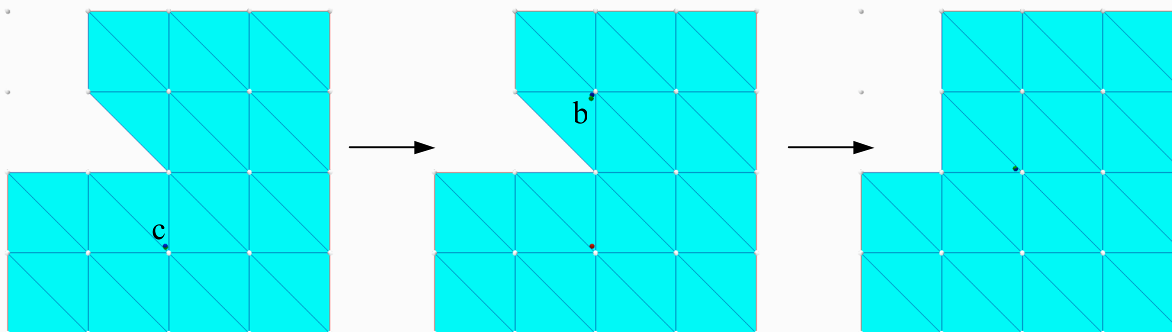
```
pt [] P = new pt[10000];          // declares an array of  vertices for the loop
int n;                             // number of vertices (initially = number of keyframes)
int f=0;                           // current frame number used for animation

void setup() { size(800, 800);  setColors();
  declarePoints();                          // creates all points P[i]
  loadPts();                                // loads n and P[0]…P[n–1] from file
  myInit();                                 // executed once to set up your animation
  }

void draw() { background(121);  myAnimate(); };  // repeated at each frame

pt S(pt A, float s, pt B) {return new pt(A.x+s*(B.x-A.x),A.y+s*(B.y-A.y)); };      // returns A+sAB
int n(int j) {  if (j==n-1) {return (0);}  else {return(j+1);}  };                  // next point in loop
int p(int j) {  if (j==0) {return (n-1);}  else {return(j-1);}  };                  // previous point in loop

void myAnimate() {                    // provide code here for running your animation (including the display of the red dot)
   if(f>n-1) f=0;                  // advance f in a cyclic manner
   fill(red);  P[f++].show(5);  // show current frame as a red disk
   }

void myInit() {                          // provide code here for setting your animation
  for (int r=0; r<=4; r++) {                 // apply 4 four-point refinement steps to P
    pt[] Q = new pt [2*n];                   // temporary array of vertices twice longer than P
    for (int i=0; i<n; i++) {                // for each edge (P[i],P[n(i)]) append 2 consecutive vertices to Q
      Q[2*i]=P(P[i]);                                                          // first a copy of P[i]
      Q[2*i+1]=P( S( S(P[p(i)],1.125,P[i]) , 0.5 , S(P[n(n(i))],1.125,P[n(i)]) ) ); }; // then the bulged mid-edge point
    n*=2;                                     // double the vertex count in P
    for (int i=0; i<n; i++) P[i].setTo(Q[i]);    // copy Q back to P overwriting previous values
    }

}  // end of myInit
```

2) **Distance**: You are given as input a vertex count n and the set of vertices P[0] ..P[n–1] of a planar polyloop (cyclic polygon). First write the Processing code **minDistToVertices**(Q) that computes the minimum distance from a point Q to the set of **vertices** of P. Then, write the Processing code **minDistToLoop**(Q) for computing the minimum distance between Q and the polyloop. Feel free to use any of the functions for which the code is provided below.

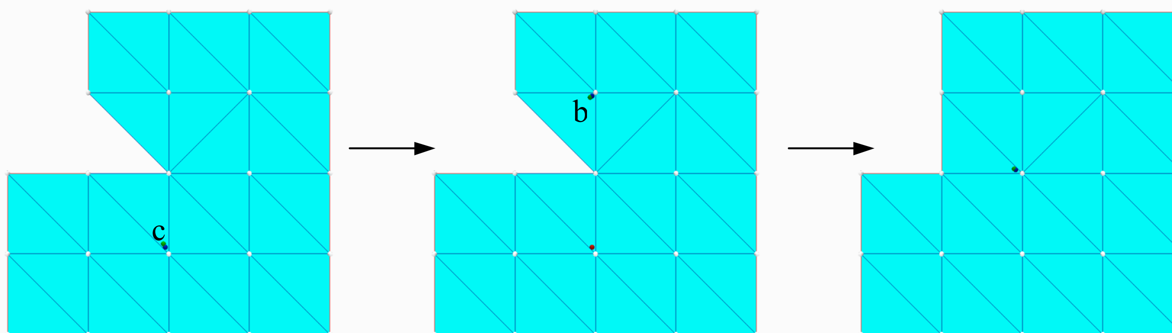
```
int n;                                  // number of points
pt [] P = new pt[10000];                // declares an array of  vertices
float d(pt P, pt Q) {return sqrt(sq(Q.x-P.x)+sq(Q.y-P.y)); };                 // ||AB||
int n(int j) {  if (j==n-1) {return (0);}  else {return(j+1);}  };            // next point in loop
int p(int j) {  if (j==0) {return (n-1);}  else {return(j-1);}  };            // previous point in loop
vec V(pt P, pt Q) {return new vec(Q.x-P.x,Q.y-P.y);};                         // PQ
vec R(vec V) {return new vec(-V.y,V.x);};                                     // V turned right 90 degrees (as seen on screen)
vec U(vec V) {float d = norm(V); if (d==0) return new vec(0,0); else return new vec(V.x/d,V.y/d);}; // V/||V||
float dot(vec U, vec V) {return U.x*V.x+U.y*V.y; };                           //U*V
```

```
float minDistToVertices(pt Q) {
  float d=d(Q,P[0]);  for (int i=1; i<n; i++) if (d(Q,P[i])<d) d = d(Q,P[i]);
  return d;
  }

float minDistToLoop(pt Q) {
  float d=minDistToVertices(Q);
  for (int i=0; i<n; i++)
     if (projectsBetween(P[i],Q,P[n(i)])  && (disToLine(Q,P[i],P[n(i)])<d)) {d=disToLine(Q,P[i],P[n(i)]);};
  return d;
  }

boolean projectsBetween(pt A, pt Q, pt B) {  // returns true if normal projection of Q on line(A,B) falls between A and B
  float q=dot(V(A,Q),V(A,B)), b=dot(V(A,B),V(A,B)) return (0<q)&&(q<b) ; }

float disToLine(pt Q, pt A, pt B) {  // returns the distance from Q to the line passing through A and B
  return abs(dot(V(A,Q),U(R(V(A,B))))); }
```

3) **Mesh**: You are given the corner table (arrays G, V, and O) of a planar triangle mesh with borders. It has nt triangles and nv vertices. You are given a corner c that has no opposite. Provide the Processing code **nextCornerAlongBorder**(c) for locating the corner b that has no opposite and that is the next corner after c along the border. Then, provide the Processing code **addEar**(c) that adds a new triangle to the mesh that is bounded by the edge opposite to corner c and the edge opposite to corner b. Its third edge is a border edge. Make sure that you correctly update the entries to G, V, and O so that the Corner Table is complete, all triangles are properly oriented, and that the opposite corner references are properly set for all corners. The figures below show two different examples. Make sure that your code will work in general (not only for these two examples). You should use only the following corner operators: n(c), which is the next clockwise corner in t(c), p(c), o(c), v(c), and b(c), which is true when c has no opposite.

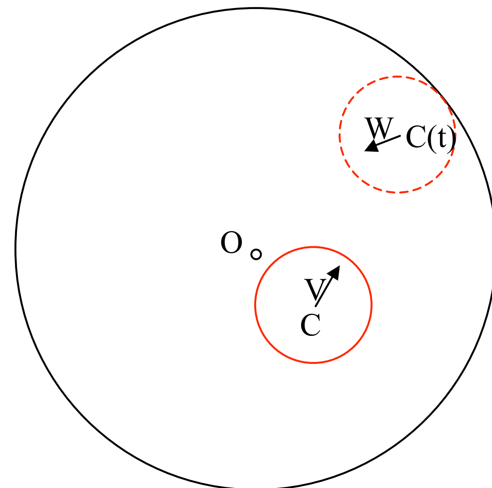


b=nextCornerAlongBorder(c) **addEar**(c)




```
int   nextCornerAlongBorder(int c) {  // computes the corner facing the next boundary edge, assumes b(c)
        int b=n(c);                   // first candidate for b. It will be kept if b(b)
        while (!b(b)) b=n(o(b));   // keep rotating b around the border vertex until if faces a border edge
        return b;
        }

void addEar(int c) {      // adds a new triangle glued to this and the next border edge
        int b = nextCornerAlongBorder(c);
        nc=3*nt;
        V[nc]=v(p(c)); O[nc]=  -1;  nc++;
        V[nc]=v(n(c)); O[nc]= b;  O[b]=nc; nc++;
        V[nc]=v(p(b)); O[nc]= c;  O[c]=nc; nc++;
        nt+=3;
        }
```

4) **Collision**: Consider a large circle with center O and radius R. A small disk of fixed radius r is trapped inside the large circle. It starts with its center at C and moves with constant velocity V. Explain how to compute the collision time t, the location C(t) of the center at the collision time, and the new velocity W after an elastic shock. Provide a sequence of geometric constructions that derive and justify the resulting formulae.



For an arbitrary time s: $C(s)=C+sV$

Collision condition: $d(O,C(s))=R-r$

Squaring it yields: $(C(s)-O)\bullet(C(s)-O)=(R-r)^2$

Substituting C(s) yields: $(C+sV-O)\bullet(C+sV-O)=(R-r)^2$

Rearranging yields: $(OC+sV)\bullet(OC+sV)=(R-r)^2$

Distributing • and rearranging yields: $V^2s^2+(2V\bullet OC)s+ OC\bullet OC-(R-r)^2=0$

Let t be the positive root of this second order equation in s (the other root must be negative if we are in the large circle).

We compute the center location at collision time as $C(t)=C+tV$.

The unit normal to both circles at that time is $N=U(V(C(t),O))$

The reflected velocity is $W=V-2(V\bullet N)N$