Polygon

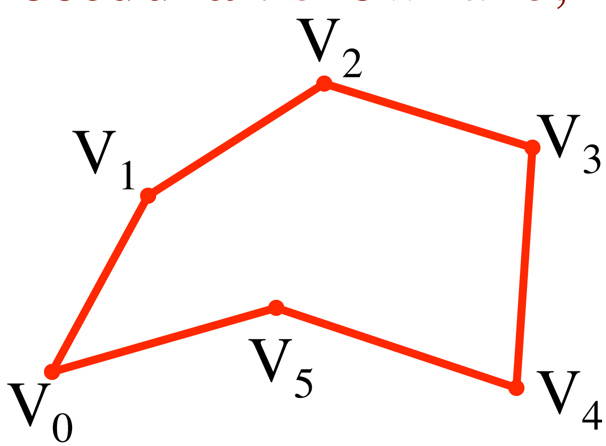
Georgia Tech College of Computing
Interactive & Intelligent Computing

- Parameterization
- Point inclusion test
- Moment
- Velocity, tangent, curvature, jerk
- Arc-length parameterization

Updated August 25, 2011

How to specify a curve?

- **Implicit** (equation): $x^2+y^2=r^2$
  - Not convenient for walking on the curve
- **Parametric** (function): $P(s) = (r \cos(s) , r \sin(s))$
- **Polyloop**: cycles of edges joining consecutive points (**vertices**)
  - Not smooth,unless you use lots of vertices
- **Subdivision**: result to which a subdivision process converges
- **Procedural**: snowflake, fractal, subdivision…

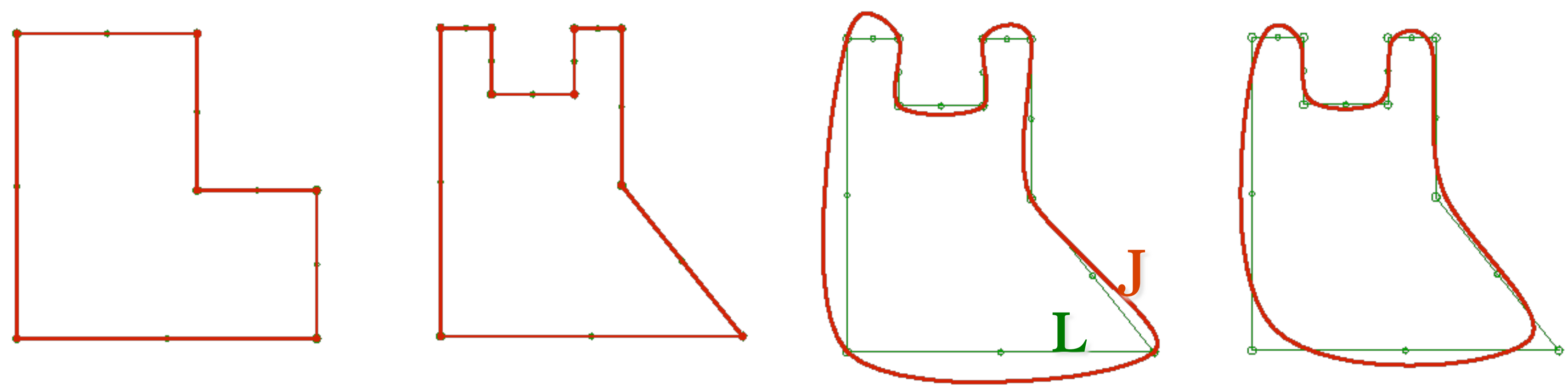


What is a polyloop L?

- A **closed loop curve** defined by a **cyclically ordered** set of **control vertices**
- It is convenient to have **next** (in) and **previous** (ip) functions for accessing them

```
int vn =6;                    // number of control vertices
pt[] P = new pt [vn];         // vertices of the control polyloop
int in(int j) {  if (j==vn-1) {return (0);}  else {return(j+1);}  };  // next vertex in control loop
int ip(int j) {  if (j==0) {return (vn-1);}  else {return(j-1);}  };  // previous vertex in control loop
```

- The user should be able to **insert**, **delete**, and **move** the control vertices…
- and get a smooth curve J **interpolating** or **approximating** the polyloop

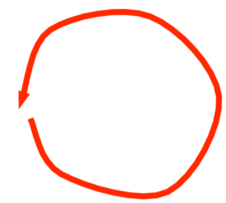


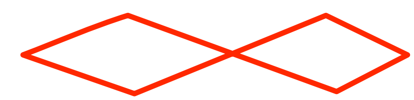
Concavity and inflection

- How to test whether corner (A,B,C) is a left turn?

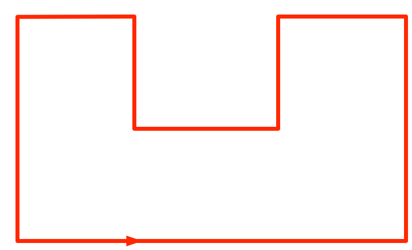
  BC•AB.left>0

- A curve is convex when all corners are left turns or all corners are right turns
  - CW: clockwise, CCW: counterclockwise
  - Convention: we **orient** the curve ccw
  - Some curves cannot be oriented (which?)





- An **inflection edge** joins a left and a right turn corner



Concave vertices?

Inflection edges?

Point-in-polygon test in 2D

- Given a polygon P and a point q in the plane of P, how would you test whether q lies inside P?

Point-in-polygon test

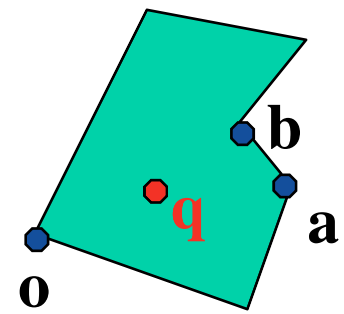
- Algorithm for testing whether point q is inside polygon P

boolean in:=false;

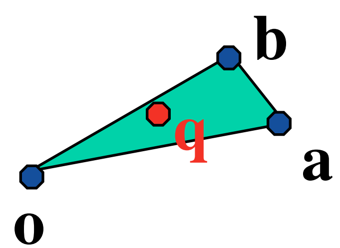
for each edge (**a**,**b**) of P { if (PinT(**q**,**a**,**b**,**o**)) in := !in; };

return in;

Point in triangle test? Nugget?

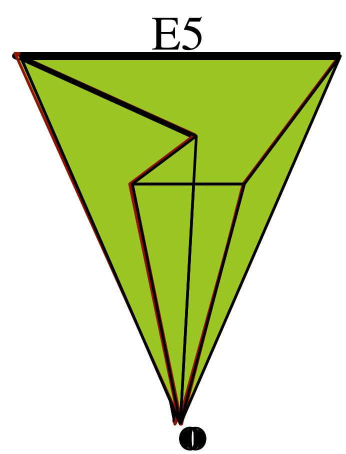


- PinT(q,a,b,o) returns true when oaq, abq, and boq are either all left turns or all right turns

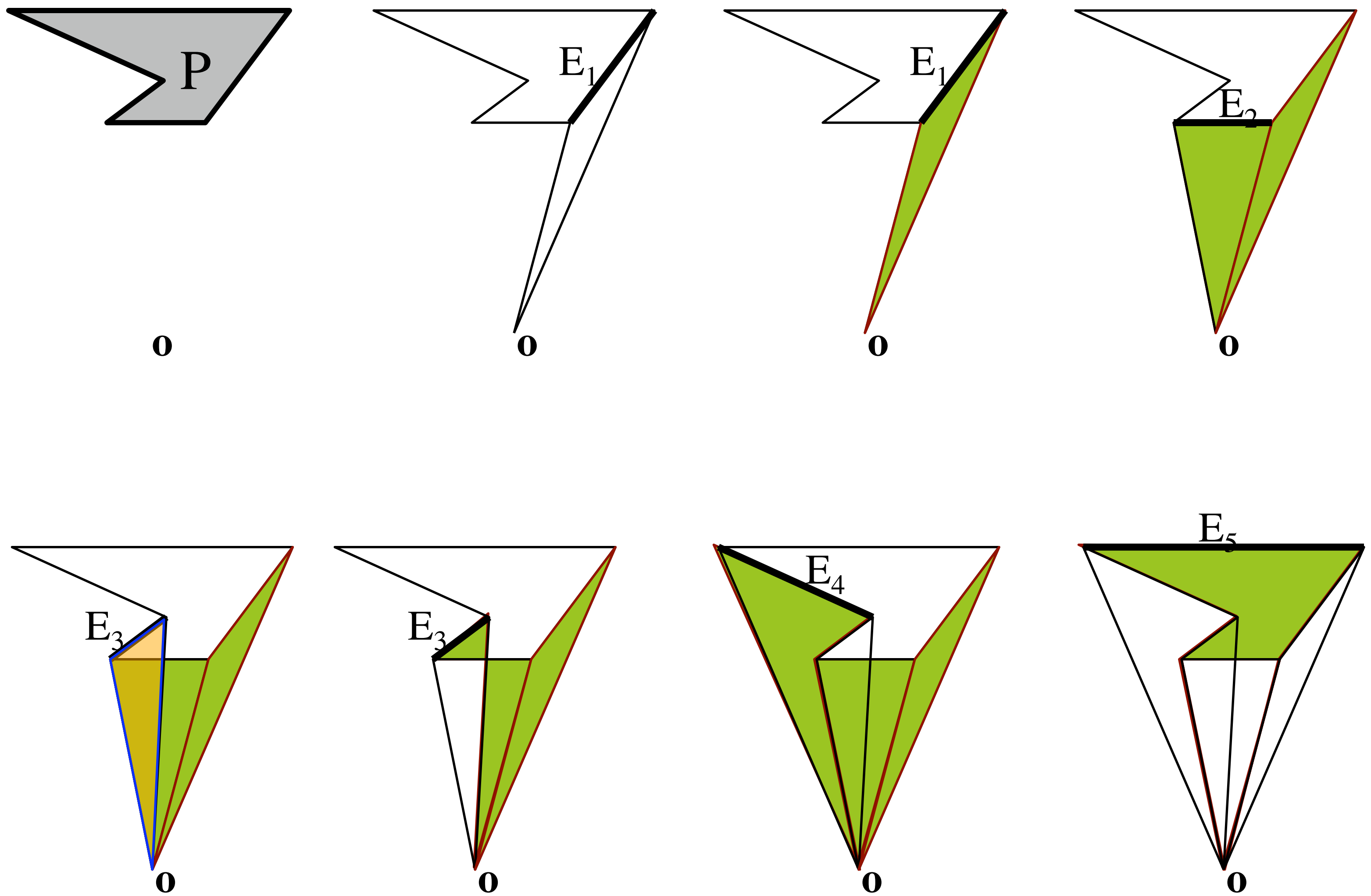


XOR shading of a polygon in 2D

- $A \otimes B$ is the set of points that lie in A or in B, but not in both
  - $A \otimes B := \{p: p \in A \mathrel{!=} p \in B\}$
- Let A, B, C, … N be primitives, then $A \otimes B \otimes C \otimes \ldots \otimes N$ is the set of points contained in an odd number of these primitives
  - $A \otimes B := \{p: (p \in A) \text{ XOR } (p \in B) \text{ XOR } (p \in C) \text{ XOR } \ldots \text{ XOR } (p \in N)\}$
- How to shade a polygon A in 2D:
  - Given a polygon A, with edges $E_1, E_2, \ldots E_n$, let T_i denote the triangle having as vertices an arbitrary origin o and the two endpoints of E_i.
  - The interior of A is $T_1 \otimes T_2 \otimes T_3 \otimes \ldots \otimes T_n$
    - *If you ignore the cracks*
  - To shade A:
    - Initialize all pixels to be background
    - Shade all the T_i using XOR
      - Toggle status of each visited pixel



Example of XOR polygon filling



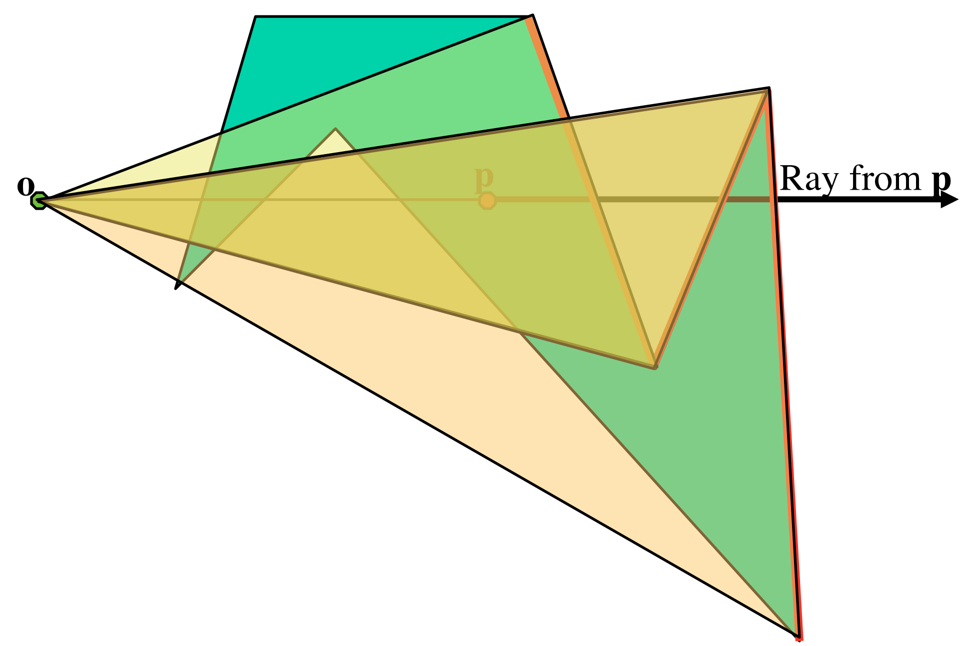
Relation to ray-casting approach?

A point p lies in a set A if a ray from p intersects the boundary of A an odd number of times

- If your ray hits a vertex or edge or is tangent to a surface, pick another ray

We do the same thing. Look at an example. Here:

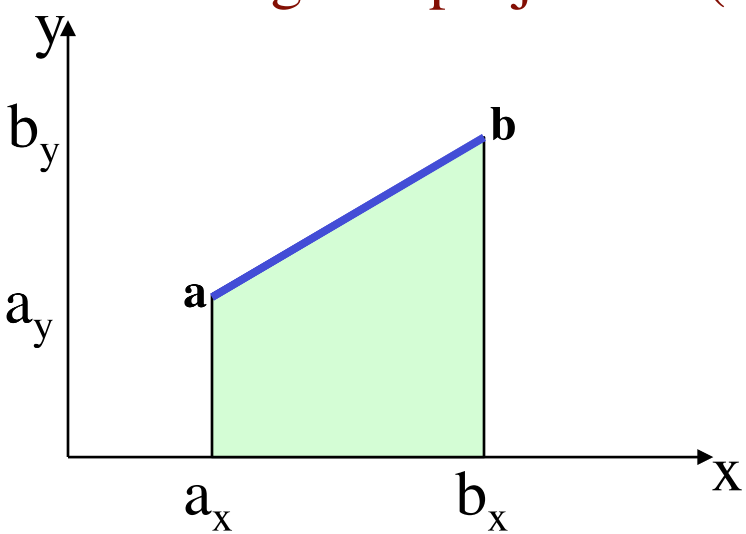
- Only edges E_1, E_2, E_3 intersect the ray from p in the opposite direction to o
- Thus only triangles T_1, T_2, T_3 contain p



p is in because it is contained in an **odd** number of triangles

Polygon area calculation: 2 methods

- Sum of signed areas of triangles, each joining an arbitrary origin o to a different edge(a,b)
  - SUM (oa⊥ob) for each edge (a,b)
    - u⊥v is the dot product of v with the the vector obtained by rotating u by 90°
- Sum of signed areas of trapezoids between each oriented edge and its orthogonal projection (shadow) on the x-axis



area(**a**,**b**):=$(a_y+b_y)(b_x-a_x)/2$

Area implementation

```
float area () {
    float A=0;
    for (int i=0; i<vn; i++) A+=trapezeArea(P[i],P[this.in(i)]);
    return(A);
    }

float trapezeArea(pt A, pt B) {return((B.x-A.x)*(B.y+A.y)/2.);}
```

Centroid of polygonal region

- Weighted sum of centroids of trapezoids
- Weight = signed area of trapezoid / total area

Centroid of trapezoid

Any line through centroid cuts shape in 2 parts with same moment

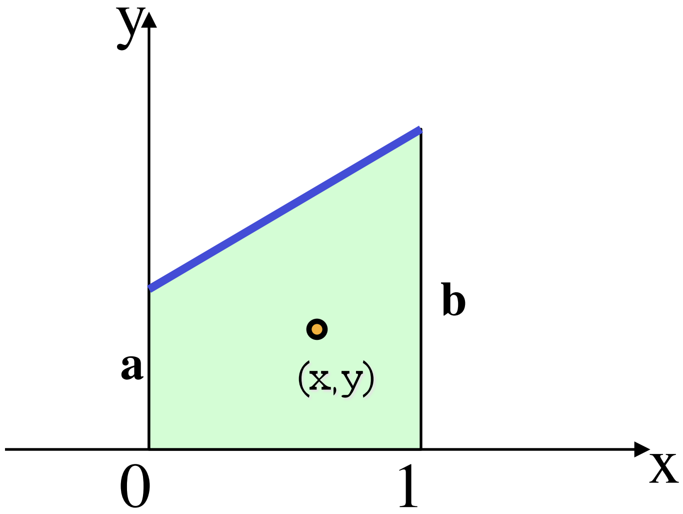
$x = {}_0\int^1 s\,y(s)ds \;/\; {}_0\int^1 y(s)ds$, with $y(s)=a+s(b-a)$

$x = {}_0\int^1 (b-a)s^2 + as\,ds \;/\; {}_0\int^1 (b-a)s + a\,ds$

$x=(a+2b)/(3(a+b))$

Similar derivation:

$y= (a^2+ab+b^2)/(3(a+b))$



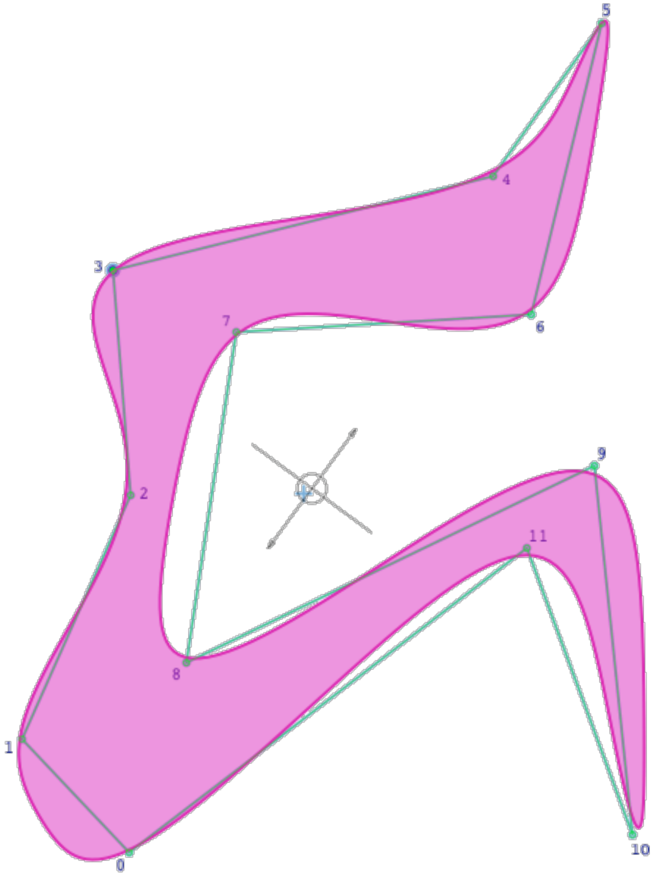
Implementation of centroid

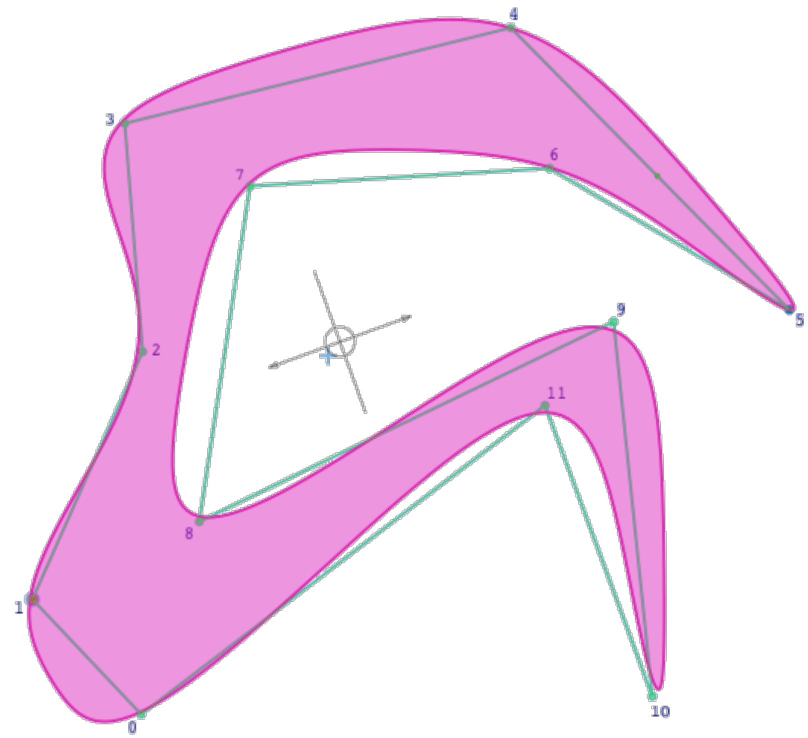
```
pt barycenter () {G.setTo(0,0);
  for (int i=0; i<vn; i++)
    G.addScaledPt(
     trapezeArea(P[i],P[this.in(i)]),trapezeCenter(P[i],P[this.in(i)]));
  G.scaleBy(1./this.area());
  return(G); }

pt trapezeCenter(pt A, pt B) {
  return(new pt(
   A.x+(B.x-A.x)*(A.y+2*B.y)/(A.y+B.y)/3., (A.y*A.y+A.y*B.y
   +B.y*B.y)/(A.y+B.y)/3.)
   ); }
```

Principal directions of polyloop

- For a set of points sampled along the curve
- Not the same as principal axes (second order inertia moments)





Implementation of principal direction

```
void showAxis() {pt G=this.barycenter(); stroke(black); G.show(10);
  float xx=0, xy=0, yy=0, px=0, py=0, mx=0, my=0;
  for (int i=0; i<vn; i++) {xx+=(P[i].x-G.x)*(P[i].x-G.x); xy+=(P[i].x-
   G.x)*(P[i].y-G.y); yy+=(P[i].y-G.y)*(P[i].y-G.y);};
  float a = atan(2*xy/(xx-yy))/2.;
  vec V= new vec(50,0); V.rotateBy(a); vec U = V.makeTurnedLeft();
  for (int i=0; i<vn; i++) {vec W=G.makeVecTo(P[i]); float vd=dot(W,V); float
   ud=dot(W,U); if(vd>0) px+=vd; else mx-=vd; if(ud>0) py+=ud; else my-
   =ud; };
  if(mx>max(px,py,my)) V.rotateBy(PI);   if(py>max(px,mx,my))
   V.rotateBy(PI/2.);  if(my>max(px,mx,py)) V.rotateBy(-PI/2.);
  strokeWeight(1); stroke(black); V.showArrowAt(G);
  stroke(black,60); V.turnLeft(); V.showAt(G); V.turnLeft(); V.showAt(G);
   V.turnLeft(); V.showAt(G);
  }
```

Questions

- Describe two techniques for testing whether a point q lies inside a polygon P
- Describe two techniques for computing the area of a polygon P
- Assume that you have a function fill_tri(a,b,c) that will visit all pixels whose center falls inside the triangle with vertices (a,b,c). Assume that each time the function visits a pixel, it toggles its status. Initially, all the status of each pixel is OFF. Explain how you could turn to ON the status of all pixels whose center falls inside a polygon P.

Closest point on an edge

- Consider Edge(A,B) and a point P in the plane
- Compute the point C of Edge(A,B) that is the closest to P
  - First compute the parameter s of the orthogonal projection H of P on Edge(A,B)
  - Then test it against [0,1] and decide

Procedure project (A,B,P) {

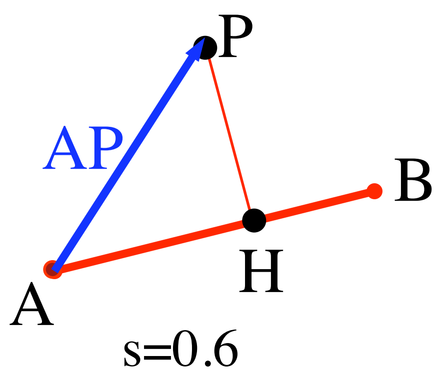
s:=AP•AB/AB•AB; # assuming A≠B

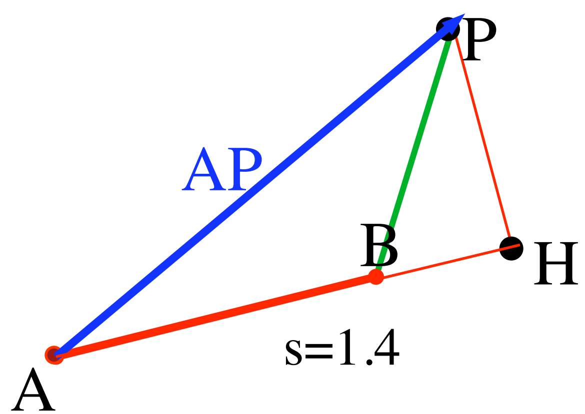
IF 0<s<1 THEN RETURN A+sAB

ELSE IF s<0

THEN RETURN A

ELSE RETURN B }





To compute closest point to P on polyloop:

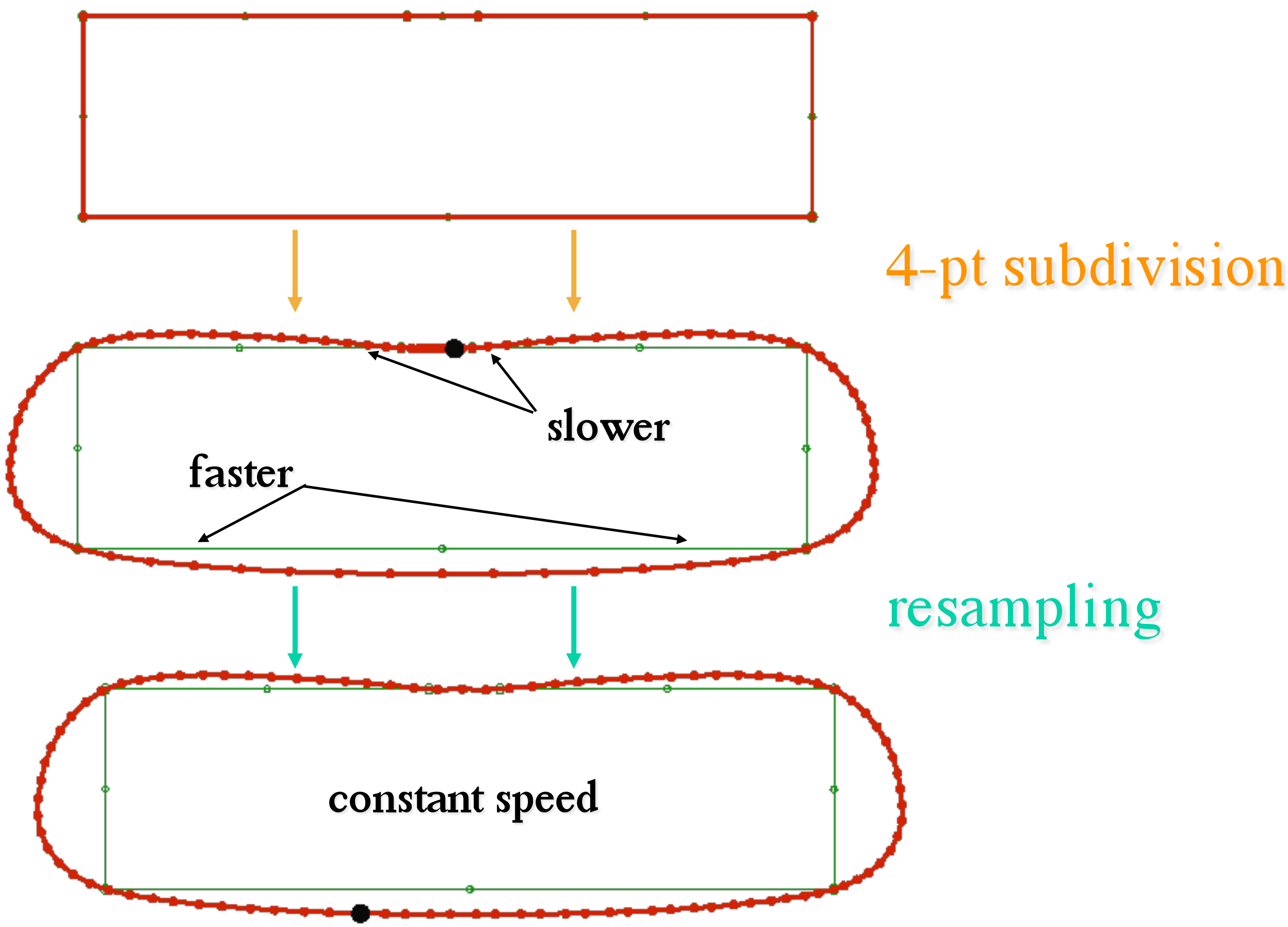
Compute it for all edges and take closest

Resampling

- You have a polyloop P path defined by sampling positions P_i
- In general, the vertices of P are not uniformly spaced
  - Their spacing may reflect the animator's desire to slow down near turns or at specific places
  - or it may be the undesired side effect of data acquisition or creation
- We may want to re-parameterize P to obtain a path P' that moves at constant speed if you use $s(P'_i, t, P'_{i+1})$ on each segment
- Then, you can warp time to impose your own speed profile, independently on the original sampling of P

Example where resampling is needed

Subdivision of a bad control polygon



Walking along an edge

- Consider an edge from point A to point B: denoted **Edge**(A,B)
- You can use a parameter, s, in [0,1] to identify any point P(s) on it

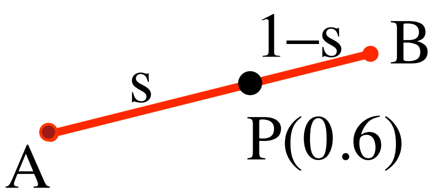
$$P(s)=A+sAB$$

  - This is the proper notation: start at A and move by s along vector AB
  - For convenience, we often use another **equivalent** notation:

$$P = A+s(B–A)$$

$$= (1–s)A+sB$$

  - Note that P(0)=A and P(1)=B



Arclength of an edge

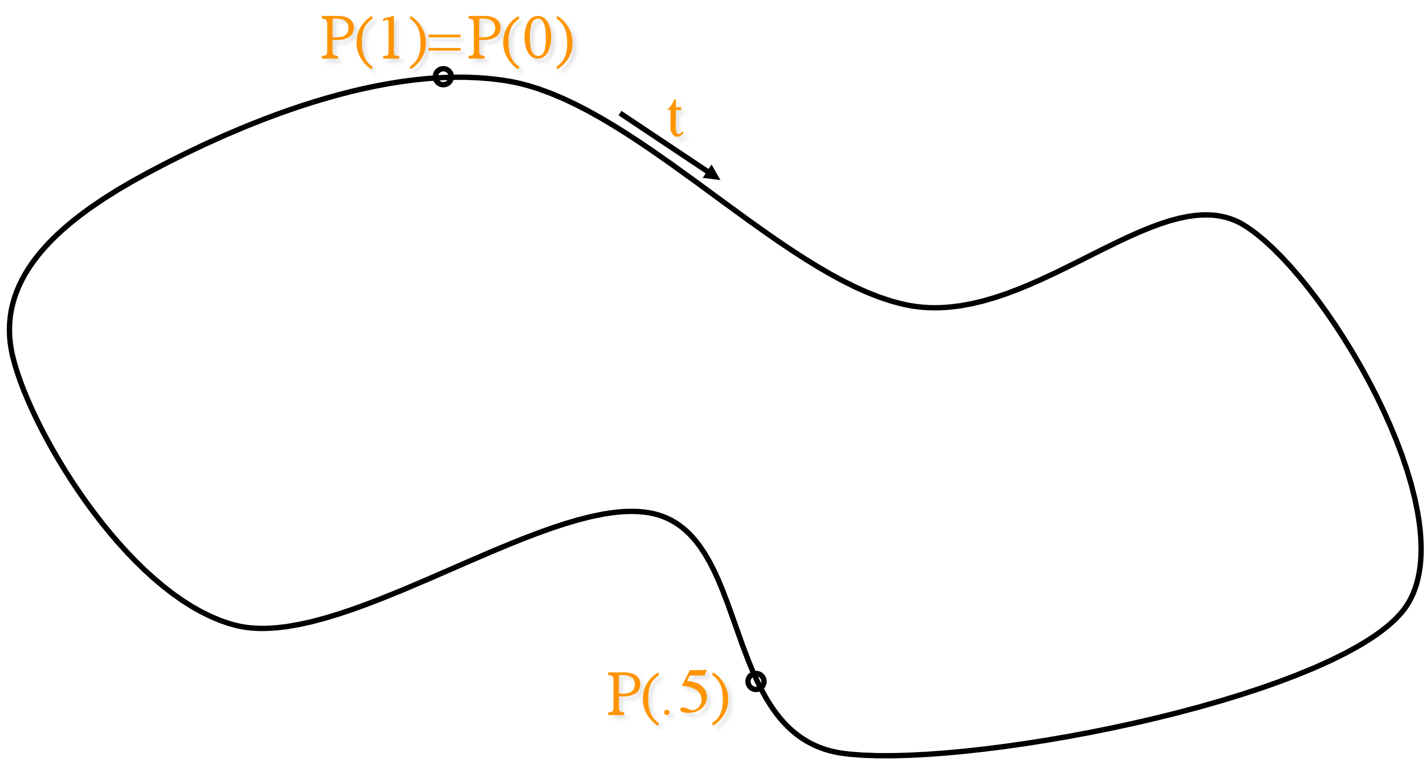
- Write the procedure L(A,B) that returns the arclength of edge (A,B)

Total arclength of a polyloop

- Write the procedure L(C) that returns the total arclength of polyloop with vertices C[i], for i = 0, 1, … n

Arclength reparameterization

- Write the procedure for computing a point on the polyloop that corresponds to parameter t in [0,1]
- Demonstrate it by animating a point at constant speed



Research challenge!!!

Given a polyloop P, place an ordered set of k points along P so that each point is at the same Euclidean (shortest) distance to its two neighbors.

Is is always possible?

Is there a deterministic algorithm?

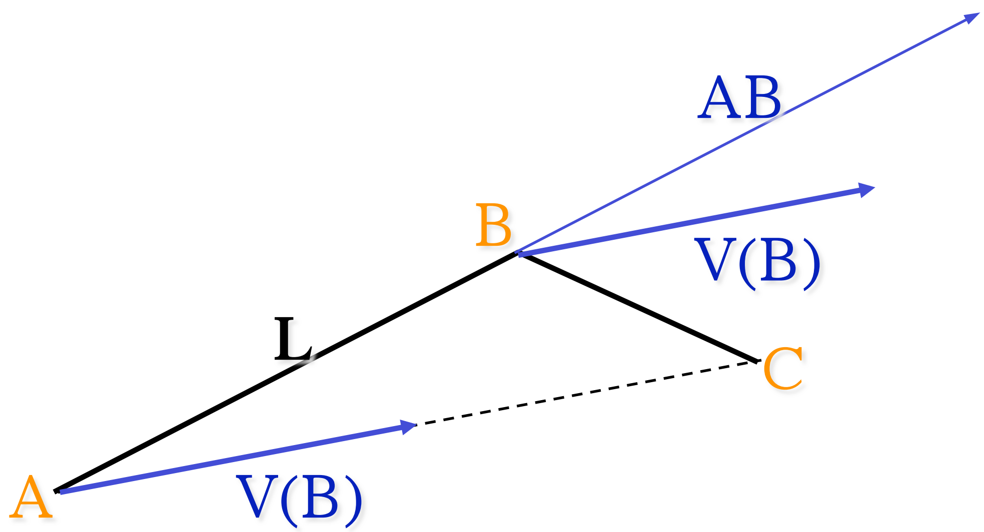
Would an iterative algorithms always converge?

Local curve analysis

- We first investigate local differential properties of a polyloop L
  - to measure forces, compare schemes, adjust speeds…
- … but these are trivial for a polyloop
  - straight line segments (zero curvature)
  - not differentiable at vertices (infinite curvature)
- So… we really want the properties of the **smooth curve** or **path** J defined by L
  - tangent, normal, acceleration, curvature, jerk
- … but, we usually do not know J
  - it may be defined as the limit of a smoothing process
- Therefore, we use **discrete estimators** for these properties

Velocity at a vertex: **V(B)=(AB+BC)/2**

- Let A,B,C,D,E… be consecutive **vertices** in L
- AB, BC,… are the **edges** of L
- J is the unknown smooth curve passing through the vertices
- When traveling along J from A to B, it takes 1 sec.
- Hence the **average velocity vector** for that segment is AB.
- Hence the velocity at B is estimated as **V(B)=(AB+BC)/2**



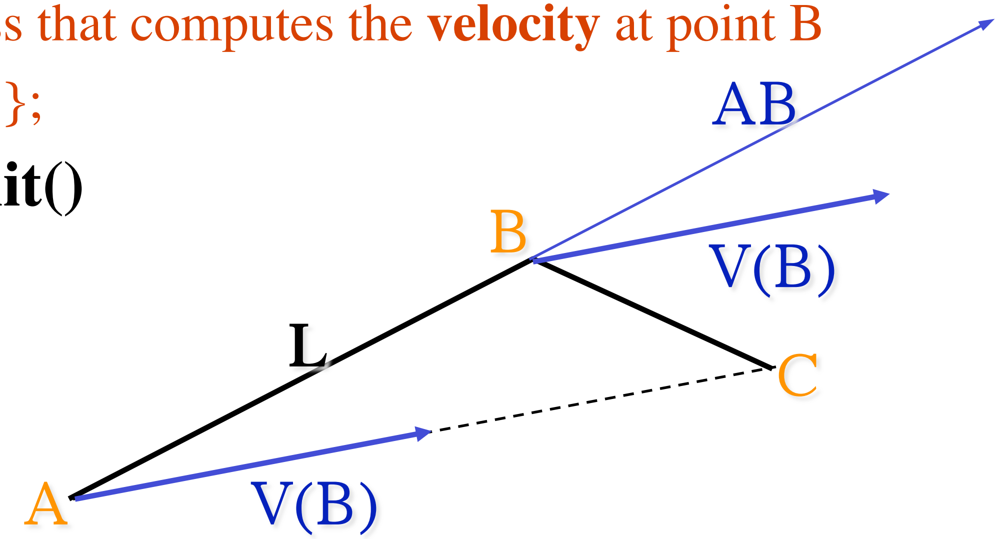
Implementing **V(B)=(AB+BC)/2**

```
class vec { float x,y;
  vec average(vec U, vec V) {return(new vec((U.x+V.x)/2,(U.y+V.y)/2)); };
  void unit() {float n=sqrt(sq(x)+sq(y)); if (n>0.000001) {x/=n; y/=n;};};
  …
class pt { float x,y;
  vec vecTo(pt P) {return(new vec(P.x-x,P.y-y)); };
```

Write a method for the pt class that computes the **velocity** at point B

vec velocity (pt A, pt C) {…};

tangent T(B) = V(B).unit()



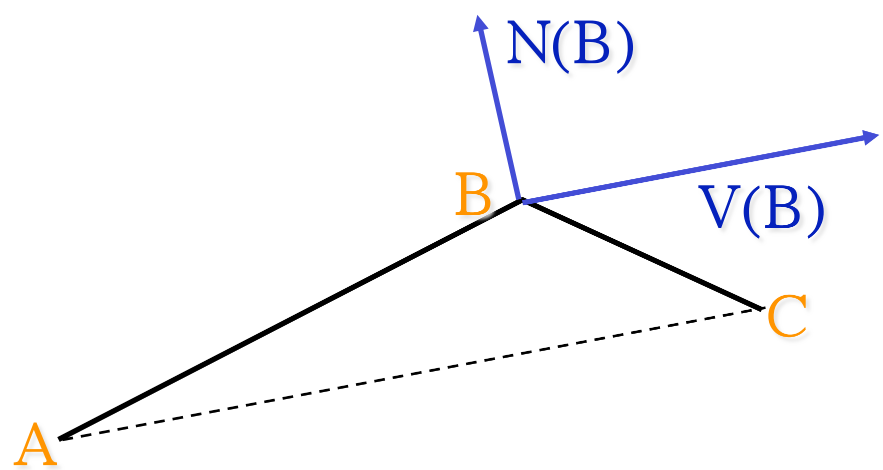
Normal at a vertex: **N(B)=AC.left().unit()**

- The normal N(B) at B is the unit vector orthogonal to AC.
- We pick the one pointing left: **N(B)=AC.left().unit()**

```
class vec { float x,y;
  void left() {float w=x; x=-y; y=w;};
  void unit() {float n=sqrt(sq(x)+sq(y)); if (n>0.000001) {x/=n; y/=n;};};
```

Write a method for the pt class that computes the **normal** at point B

```
vec normal (pt A, pt C) {…};
```

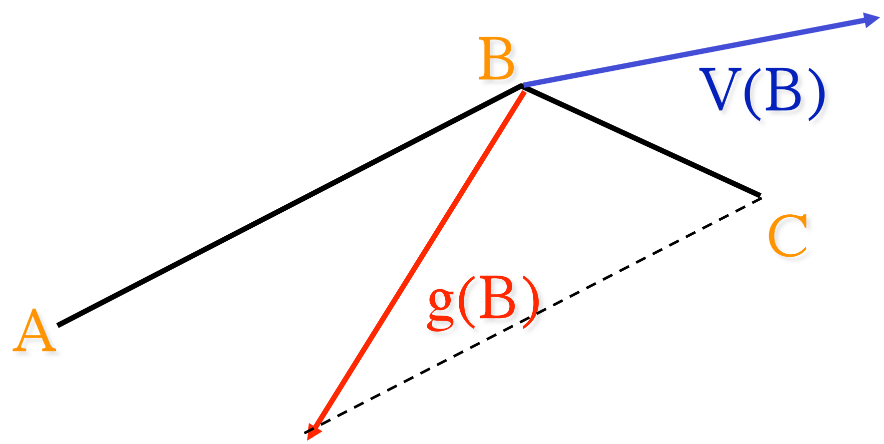


Acceleration at a vertex: **g(B)=BC–AB**

- The acceleration at B is the velocity change **g(B)=BC–AB**
- It is useful for computing forces during animation (dynamics)

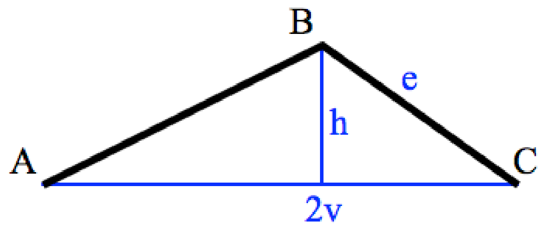
Write a method for the pt class that computes the **acceleration** at point B

vec acceleration (pt A, pt C) {…};



Radius of curvature: **r(B)=V²/(2AB•N(B))**

- The radius r(B) of curvature (sharpness of turn in the path) at B could be estimated as the radius of the circle through A, B, and C, but this approach yields unexpected results when the angle at b is acute.
- I prefer to use the **parabolic curvature**, **$r(B)=V^2/(2h)$,** where V is the magnitude of V(B) and where h is the distance from B to the Line(A,C).
- h=AB•N(B) is the normal component of g(B)
- It measures the centrifugal force
- **Curvature** = 1/r(B)



Write a method for the pt class that computes the **r(B)** at point B

vec radiusOfCurvature (pt A, pt C) {…};

Jerk: j(B,C)=AD+3CB

It is the change of acceleration between B and C. It measures the change in the force felt by a person traveling along the curve

j(B,C) = g(C)–g(B)

= (CD–BC)–(BC–AB) = CD+CB+CB+AB

= AB+BC+CD + CB+CB+CB = AD+3CB

To show the second degree discontinuities in the curve,

draw the normal component of j(B,C) at (B+C)/2