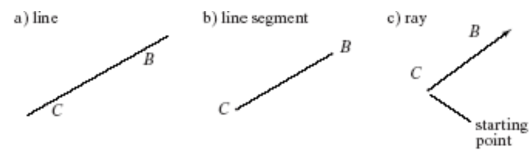
Computer Graphics

Vector Tools for Graphics
Sections 4.4-4.8

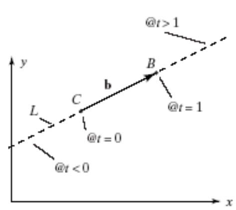
Representing Lines

- A line passes through 2 points and is infinitely long.
- A line segment has 2 endpoints.
- A ray has a single endpoint.



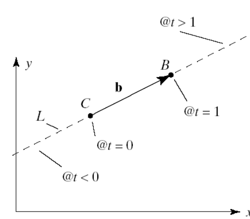
Representing Lines (2) --parametric form

- There are 2 useful line representations:
- Parametric form: we have 2 points, B and C, on the line. P(x, y) is on the line when P = C + **b**t, where **b** = B – C.
  - $0 \le t \le 1$: line segment;
  - $-\infty \le t \le \infty$: line;
  - $-\infty \le t \le 0$ or $0 \le t \le \infty$: ray.



Representing Lines (3) -- parametric form

- As *t* varies so does the position of *L*(*t*) along the line. (Let *t* be time.)
- If *t* = 0, *L*(0) = *C* so at *t* = 0 we are at point *C.*
- At *t* = 1, *L*(1) = *C* + (*B* - *C*) = *B*.
- If *t* > 1 this point lies somewhere on the opposite side of *B* from *C*, and when *t* < 0 it lies on the opposite side of *C* from *B*.



Representing Lines (4)

- $L(t)$ lies fraction t of the way between C and B when t lies between 0 and 1.
- When t = 1/2 the point $L(0.5)$ is the **midpoint** between C and B, and when t = 0.3 the point $L(0.3)$ is 30% of the way from C to B:

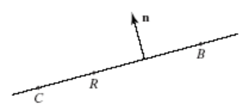
  $|L(t) - C| = |\mathbf{b}|\ |t|$ and $|B - C| = |\mathbf{b}|$,

  so the value of $|t|$ is the ratio of the distances $|L(t) - C|$ to $|B - C|$.

Representing Lines (5) – point normal form

- Point-normal form of a line consists of any point, **R**, on the line and a vector, **n**, perpendicular to the line.

  $\mathbf{n} \cdot (R - C) = 0$



  – Given B and C on the line, **b** = B – C gives $\mathbf{b}^{\perp} = \mathbf{n}$, which is perpendicular to R – C. (R is any point (x, y) on the line.)

  – The equation is $\mathbf{n} \cdot (R - C) = 0$.

Changing Representations -- point normal form

- From $f\,x + g\,y = 1$ (implicit form) to point-normal form:
  – Example, for 2x+6y=3; (2, 6) is its normal
- From point normal form:

  $\mathbf{n} \cdot (R - C) = 0$

  to parametric form:

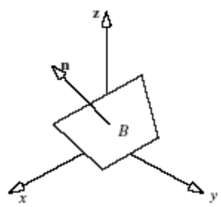
  $L(t) = C + \mathbf{n}^{\perp} t$

Practice Question

- For a line L passes through points C(3, 4) and B(5, -2), what is the point normal form for this line?
- For a line 3x+5y=11, what is the point normal form of this line?

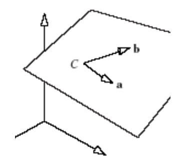
Representing Planes: Point-Normal Form

- Point-normal form: $\mathbf{n}\cdot(P - B) = 0$; where B is a given point on the plane, and $P = (x, y, z, 1)^T$.



Planes: Parametric Form

- A plane can be infinite in 2 directions, semi-infinite, or finite.
  - Parametric form: requires 3 non-collinear points on the plane, A, B, and C.
  - $P(s, t) = C + s\mathbf{a} + t\mathbf{b}$, where $\mathbf{a} = A - C$ and $\mathbf{b} = B - C$.
  - $-\infty \le s \le \infty$ and $-\infty \le t \le \infty$: infinite plane.
  - $0 \le s \le 1$ and $0 \le t \le 1$: a finite plane, or patch.



Planes: Parametric Form (2)

- We can rewrite

  $P(s,t) = C + s\mathbf{a} + t\mathbf{b}$

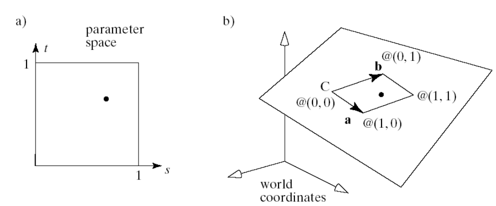
  where $\mathbf{a} = A - C$ and $\mathbf{b} = B - C$, as an affine combination of points:

  $P(s, t) = s\,A + t\,B + (1 - s - t)C$
- Compare it with the parametric form of a line:

  $L(t) = tB + (1-t)A$

Planes: Parametric Form (3)

- The figure shows the available range of *s* and *t* as a square in **parameter space**, and the patch that results from this restriction in object space.



Patches

- Mapping textures onto faces involves finding a mapping from a portion of parameter space onto object space, as we shall see later.
- Each point (s, t) in parameter space corresponds to one 3D point in the patch $P(s, t) = C + \mathbf{a}s + \mathbf{b}t$.
- The patch is a parallelogram whose corners correspond to the four corners of parameter space and are situated at :

  $P(0, 0) = C$

  $P(1, 0) = C + \mathbf{a}$

  $P(0, 1) = C + \mathbf{b}$

  $P(1, 1) = C + \mathbf{a} + \mathbf{b}$

Patches (2)

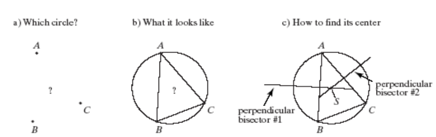
- The vectors **a** and **b** determine both the size, the shape, and the orientation of the patch.
  - If **a** and **b** are perpendicular, the grid will become rectangular.
  - If in addition **a** and **b** have the same length, the grid will become square.
- Changing C just translates the patch without changing its size, shape or orientation.

Practice Question

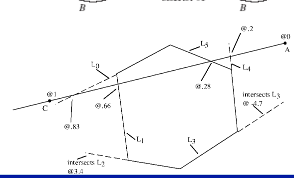
- Find **a**, **b** and **C** that create a square patch of length 4 on a side centered at the origin and parallel to the x-z plane.

Finding the Intersection of 2 Line Segments

- Find out whether two line segments intersect, and where they intersect has many useful applications in CG



(a) Draw a circle Based on 3 points

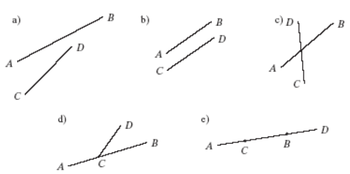


(b) Clipping line against polygon

Clipping polygon against window

Finding the Intersection of 2 Line Segments

- For two line segments, they can miss each other (a and b), overlap in one point (c and d), or even overlap over some region (e). They may or may not be parallel.



Intersection of 2 Line Segments (2)

- Every line segment has a **parent line**, the infinite line of which it is a part. Unless two parent lines are parallel, they will intersect at some point in 2D. We locate this point.
- Using parametric representations for each of the line segments in question, call *AB* the segment from *A* to *B*. Then $AB(t) = A + \mathbf{b}\, t$, where for convenience we define $\mathbf{b} = B - A$.

Middle Tennessee State University

Intersection of 2 Line Segments (3)

- For two line segments AB and CD:

  $AB(t) = A + \mathbf{b}t,\ CD(u) = C + \mathbf{d}u$

  where $\mathbf{b} = B - A$ and $\mathbf{d} = C - D$.

- At the intersection, they share the single point of intersection at time t for AB, and u for CD:

  $A + \mathbf{b}t = C + \mathbf{d}u$

  or

  $\mathbf{b}t = \mathbf{c} + \mathbf{d}u$

  with $\mathbf{c} = C - A$.

Middle Tennessee State University

Intersection of 2 Line Segments (4)

- Taking dot product with $\mathbf{d}^{\perp}$ gives $\mathbf{b}\cdot\mathbf{d}^{\perp}\ t = \mathbf{c}\cdot\mathbf{d}^{\perp}$.
- Taking dot product with $\mathbf{b}^{\perp}$ gives $-\ \mathbf{c}\cdot\mathbf{b}^{\perp} = \mathbf{d}\cdot\mathbf{b}^{\perp}\ u$.

  $t = \mathbf{c}\cdot\mathbf{d}^{\perp} / \mathbf{b}\cdot\mathbf{d}^{\perp}$ and $u = -\ \mathbf{c}\cdot\mathbf{b}^{\perp} / \mathbf{d}\cdot\mathbf{b}^{\perp}$
- Case1: $\mathbf{b}\cdot\mathbf{d}^{\perp} = 0$ means $\mathbf{d}\cdot\mathbf{b}^{\perp} = 0$ and the lines are either the same line or parallel lines. There is no intersection.
- Case 2: $\mathbf{b}\cdot\mathbf{d}^{\perp} \neq 0$, intersect at time t for AB, u for CD
- In this case, the line segments intersect if and only if $0 \le t \le 1$ and $0 \le u \le 1$, at $P = A + \mathbf{b}(\mathbf{c}\cdot\mathbf{d}^{\perp} / \mathbf{b}\cdot\mathbf{d}^{\perp})$.

Middle Tennessee State University

Practice Questions

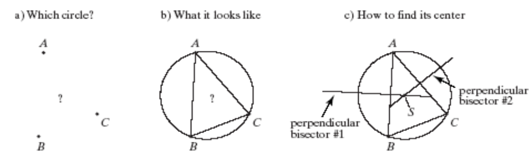
- For the following segment pairs, do they intersect? If so, when? and where?

(1) A=(1, 4), B=(7, ½)   and C=(7/2, 5/2), D=(7, 5)

(2) A=(1, 4), B=(7, ½) and C(5, 0), D=(0, 7)

Finding A Circle through 3 Points

- We want to find the center and the radius of the circle.
  - The 3 points make a triangle, and the center S is where the perpendicular bisectors of two of the sides of the triangle meet.
  - The radius is r = |A – S|.



Circle through 3 Points (2)

- The perpendicular bisector passes through the midpoint

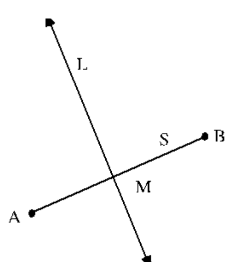
  M = ½ (A + B)

  of the line AB, in the direction (B – A)$^{\perp}$.
- Let **a** = B – A,

  M= *A* + **a** / 2;

  direction perpendicular to *AB*: **a**$^{\perp}$.
- The perpendicular bisector of AB is:

  P= *A* + a / 2 + a$^{\perp}$ *t*



Circle through 3 Points (3)

– Similarly, let **c** = A-C,

the perpendicular bisector of AC is:

Q = *A* + c / 2 + c$^{\perp}$ *u*

(using parameter *u)*

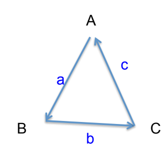
– Point *S* lies where these two perpendicular bisectors meet, at the solution of

a$^{\perp}$*t* = b / 2 + c$^{\perp}$ *u*

(b = C - B, and **a** + **b** = - **c**).

Solve t:

$$t = \frac{\frac{b}{2} \bullet c}{c \bullet a^{\perp}}$$



Circle through 3 Points (3)

– To find Center S:

$$S = A + \frac{a}{2} + a^{\perp}t$$
$$= A + \frac{1}{2}\left(a + \frac{b \bullet c}{c \bullet a^{\perp}} a^{\perp}\right)$$

– Radius R = |S – A|

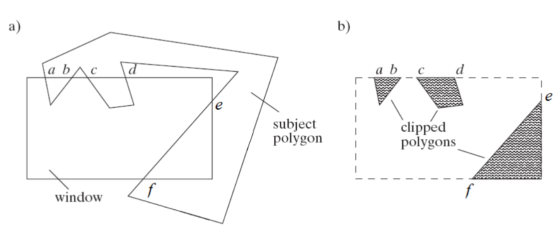
$$radius = \frac{|a|}{2}\sqrt{\left(\frac{b \bullet c}{a^{\perp} \bullet c}\right)^2 + 1}$$

Practice Question

1. Find the perpendicular bisector L of the segment S having endpoints A=(3, 5) and B=(9, 3)
2. Let the three points of a triangle equal to A=(6, 7), B=(0, -1), and C=(6, -1). Compute the center and the radius of the excircle for these three points

Clipping a Polygon

- "To clip a polygon against a window" means to find which part of the polygon is inside the window and thus will be drawn.

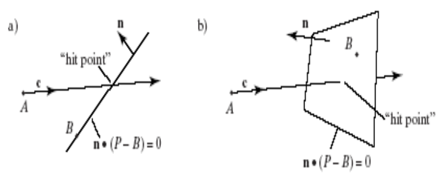


Clipping a Polygon (2)

- In Chapter 3, we looked at Cohen-Sutherland clipping of lines in a rectangular window, which involves the intersection of 2 lines.
- Here we will focus the Cyrus-Beck clipping algorithm, which clips polygons against lines and planes.

Intersections of Lines and Planes

- Intersections of a line and a line or plane are used in ray-tracing and 3D clipping: we want to find the "hit point".



Intersections of Lines and Planes (2)

- Suppose the ray hits at $t = t_{hit}$, the **hit time**.
- At this value of t the ray and line or plane must have the same coordinates, so $A + \mathbf{c}\, t_{hit}$ must satisfy the equation of the point normal form for the line or plane, $\mathbf{n}\cdot(P - B) = 0$.
- When the ray intersects (hits) the line or plane, $P = A + \mathbf{c}\, t_{hit}$, giving

$$\mathbf{n}\cdot(A + \mathbf{c}\, t_{hit} - B) = 0.$$

$$t_{hit} = \mathbf{n}\cdot(B - A)/\mathbf{n}\cdot\mathbf{c}, \quad \text{if } \mathbf{n}\cdot\mathbf{c} \neq 0.$$

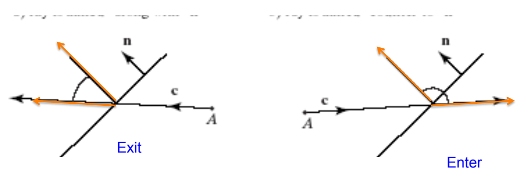
Intersections of Lines and Planes (3)

- To find the hit point P_{hit}, substitute t_{hit} into the representation of the ray:

$$P_{hit} = A + \mathbf{c}t_{hit} = A + \mathbf{c}(\mathbf{n}\cdot(B - A)/\mathbf{n}\cdot\mathbf{c}).$$

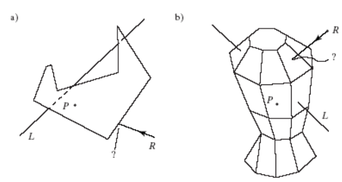
Direction of Ray

- If $\mathbf{n}\cdot\mathbf{c} = 0$, the ray is parallel to the line.
- If $\mathbf{n}\cdot\mathbf{c} > 0$, $\mathbf{c}$ and $\mathbf{n}$ make an angle of less than 90° with each other.
- If $\mathbf{n}\cdot\mathbf{c} < 0$, $\mathbf{c}$ and $\mathbf{n}$ make an angle of more than 90° with each other.



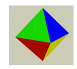
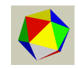
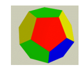
Polygon Intersection Problems

- Is a given point inside or outside the polygon?
- Where does a given ray first intersect the polygon?
- Which part of a given line L lies inside the object, and which outside?

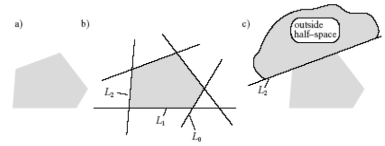


Polygon Intersection Problems (2)

- Convex Polygons and Polyhedra: a simple case

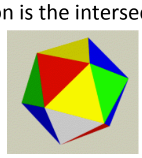
  

- They are described by a set of bounding lines/planes, and the entire polygon/polyhedron is on one side of the line/plane.



Polygon Intersection Problems (3)

- The line/plane divides space into two halves: the outside space, which shares no points with the polyhedron, and the inside half space, where the polyhedron lies.
- The polyhedron is the intersection of all the inside half spaces.



Ray Intersection Problem

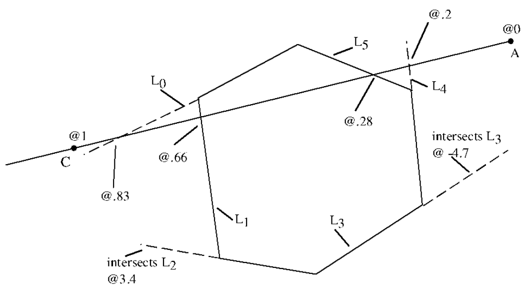
- Where does the ray $A + \mathbf{c}t$ hit convex polygon P?
- Because of convexity, each line hits the polygon twice: going in, and coming out.
- For a line A + **c**t, find its intersections with each of the boundary lines:

$$t_{hit} = \mathbf{n}\cdot(B - A)/\mathbf{n}\cdot\mathbf{c}, \text{ if } \mathbf{n}\cdot\mathbf{c} \neq 0.$$

  - If $\mathbf{n}\cdot\mathbf{c} > 0$, the line is exiting the polygon. Set $t_{out} = \min(t_{hit}, t_{out})$.
  - If $\mathbf{n}\cdot\mathbf{c} < 0$, the line is entering the polygon. Set $t_{in} = \max(t_{hit}, t_{in})$.
  - If $t_{in} > t_{out}$ the ray misses the polygon entirely. Otherwise, the ray is inside the polygon during $[t_{in}, t_{out}]$.

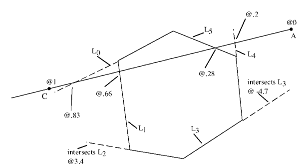
How to compute n for each side of a polygon?

Example



Example (2)

- The sequence of updates to T_{in} and T_{out} as the various line intersections are tested.



Line tested	T_{in}	T_{out}
0	0	0.83
1	0	0.66
2	0	0.66
3	0	0.66
4	0.2	0.66
5	0.28	0.66

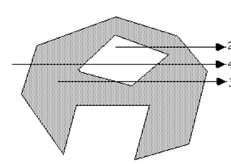
Practice Question

1. Given: A=(2, -4, 3), c=(4, 0, -4), n=(6, 9, 9), B=(-7, 2, 7), find when and where the ray A+ct hits the plane: $n \bullet (P - B) = 0$

2. Find the point where the ray (2, 4, 2)+(-1, 4, 2)t hits the plane 3x-2y+z=0.

Inside-Outside Tests

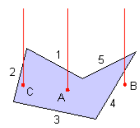
- Given an arbitrary polygon, and a point P:
  How to determine whether the point P is inside or outside the polygon?



Middle Tennessee State University

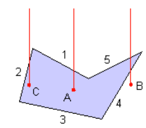
Inside-Outside Tests

- Is point P inside or outside the polygon?
  - Form a vector **u** = D – P, where D is any point outside the polygon, and **u** intersects no vertices of the polygon.
  - Create **n** normal to **u**. Let w = 0 be an int
  - Let **E** be the vector along an edge crossed by **u**. Calculate d = **E·n**:
    - if d > 0,(exit polygon case), add 1 to w;
    - if d < 0, (ray enter polygon case), subtract 1 from w.
  - Repeat for each edge crossed by **u**. If at the end, w = 0 (even number of intersects), P is outside; else (odd number of intersects) P is inside.



Inside-Outside Tests

- Is point P inside or outside the polygon?
  - Form a vector **u** = D – P, where D is any point outside the polygon, and **u** intersects no vertices of the polygon.
  - Create **n** normal to **u**. Let w = 0 be an int (the winding number.)
  - Let **E** be the vector along an edge crossed by **u**. Calculate d = **E·n**:
    - if d > 0,(exit polygon case), add 1 to w;
    - if d < 0, (ray enter polygon case), subtract 1 from w.
  - Repeat for each edge crossed by **u**. If at the end, w = 0 (even number of intersects), P is outside; else (odd number of intersects) P is inside.



Inside-Outside Tests (2)

- Equivalent method: create **u** as before.
  - Calculate t_{in}, t_{out}, and t_P (time **u** reaches P).
  - If $t_{in} \le t_P \le t_{out}$, P is inside; else it is outside.

Cyrus-Beck Clipping

- Cyrus-Beck clipping clips a line segment against any convex polygon P:
- int CyrusBeckClip(Line& seg, LineList& L) uses parameters seg (the line segment to be clipped, which contains the first and second endpoints named seg.first and seg.second), and the list L of bounding lines of the polygon.
- It clips seg against each line in *L*, and places the clipped segment back in seg. (This is why seg must be passed by reference.)
- The routine returns 0 if no part of the segment lies in *P* or 1 if some part of the segment does lie in *P*.

Cyrus-Beck Pseudo-code (2D)

- Variables *numer* and *denom* hold the numerator and denominator for t_{hit}:
  - Numer = **n**·(B – A),
  - denom = **n·c**

```
int CyrusBeckClip(LineSegment& seg, LineList& L)
{
   double numer, denom;
   double tIn = 0.0, tOut = 1.0;
   Vector2 c, tmp;
   <form vector: c = seg.second - seg.first>
```


Cyrus-Beck Pseudo-code (2)

```
for (int i = 0; i < L.num; i++) // chop at each bounding line
{
        <form vector tmp = L.line[i].pt - first>
        numer = dot(L.line[i].norm, tmp);
        denom = dot(L.line[i].norm, c);

        if (!chopCI(numer, denom, tIn, tOut))
            return 0;
        // early out
```


Cyrus-Beck Pseudo-code (3)

```
   // adjust the endpoints of the segment; do second one 1st.

   // second endpoint was altered
   if (tOut < 1.0 )
   {
        seg.second.x  = seg.first.x + c.x * tOut;
        seg.second.y  = seg.first.y + c.y * tOut;
   }
   if (tIn > 0.0)  // first endpoint was altered
   {
        seg.first.x  = seg.first.x + c.x * tIn;
        seg.first.y  = seg.first.y + c.y * tIn;
   }

   return 1; // some segment survives
}
```


Cyrus-Beck Pseudo-code (4)

- The routine chopCI() uses numer and denom to calculate the hit time at which the ray hits a bounding line, determines whether the ray is entering or exiting the polygon, and chops off the piece of the candidate interval CI that is thereby found to be outside the polygon.

Cyrus-Beck Pseudo-code (5)

```
int chopCI(double& tIn, double& tOut, double numer,
   double denom)
{  double tHit;
   if (denom < 0)          // ray is entering
   {
        tHit = numer / denom;
        if (tHit > tOut) return 0;  // early out
        else if (tHit > tIn) tIn = tHit; // take larger t
   }
```


Cyrus-Beck Pseudo-code (6)

```
   else if (denom > 0)        // ray is exiting
   {
    tHit = numer / denom;
    if(tHit < tIn) return 0;     // early out
    if(tHit < tOut) tOut = tHit; // take smaller t
   }
   else           // denom is 0: ray is parallel
    if (numer <= 0) return 0;   // missed the line
   return 1;      // CI is still non-empty
}
```


3D Cyrus-Beck Clipping

- The Cyrus Beck clipping algorithm works in three dimensions in exactly the same way.
- In 3D the edges of the window become planes defining a convex region in three dimensions, and the line segment is a line in 3D space.
- ChopCI() needs no changes at all (since it uses only the values of dot products).
- The data types in CyrusBeckClip() must of course be extended to 3D types, and when the endpoints of the line are adjusted the z-component must be adjusted as well.