Computer Graphics

Vector Tools for Graphics
(Ch4, sec 1-3)

Introduction

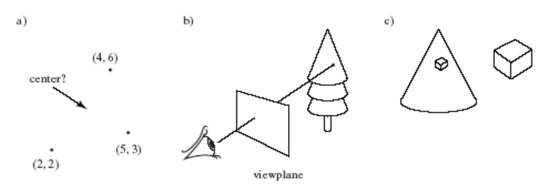
- In computer graphics, we work with objects defined in a three dimensional world (with 2D objects and worlds being just special cases).
- All objects to be drawn, and the cameras used to draw them, have shape, position, and orientation.
- We must write computer programs that somehow describe these objects, and describe how light bounces around illuminating them, so that the final pixel values on the display can be computed.

Introduction (2)

- The two fundamental sets of tools that come to our aid in graphics are *vector analysis* (Ch. 4) and *transformations* (Ch. 5).
- We develop methods to describe various geometric objects, and we learn how to convert geometric ideas to numbers.
- This provides a collection of crucial algorithms that we can use in graphics programs.

Easy Problems for Vectors

- Where is the center of the circle through the 3 points? What image shape appears on the viewplane, and where? Where does the reflection of the cube appear on the shiny cone, and what is the exact shape of the reflection?

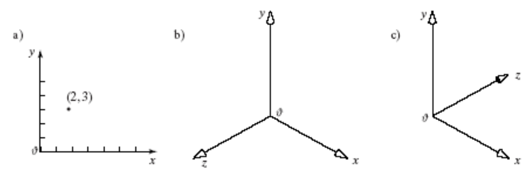


Vectors

- Vectors provide easy ways of solving some tough problems.
- A vector has length and direction, but not position (relative to a coordinate system). It can be moved anywhere.
- A point has position but not length and direction (relative to a coordinate system).
- A scalar has only size (a number).

Basics of Points and Vectors

- All points and vectors are defined relative to some coordinate system. Shown below are a 2D coordinate system and a right- and a left-handed 3-D coordinate system.

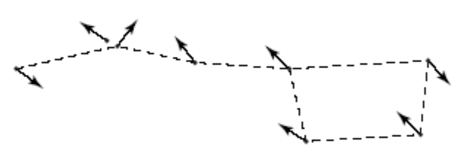


Left and Right Handedness

- In a 3D system, using your right hand, curl your fingers around going from the x-axis to the y-axis. Your thumb is at right angles to your fingers.
  - If your thumb points along the direction of the z-axis, the system is right handed.
  - If your thumb points opposite to the direction of the z-axis, the system is left handed.

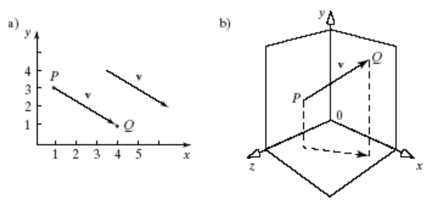
4.2: Review of Vectors

- Vectors are drawn as arrows of a certain length pointing in a certain direction.
- A vector is a *displacement* from one point to another. Shown below are displacements of the stars in the Big Dipper over the next 50,000 years.



Vectors and Coordinate Systems

- A vector v between points P = (1, 3) and Q = (4, 1), with components of (3, -2), calculated by subtracting the coordinates individually (Q – P).
- To "go" from P to Q, we move down by 2 and right by 3. Since v has no position, the two arrows labeled v are the same vector. The 3D case is also shown.



Vector Operations

- The difference between 2 points is a vector: **v** = Q – P.
- The sum of a point and a vector is a point: P + **v** = Q.
- We represent an n-dimensional vector by an n-tuple of its components, e.g. **v** = (v_x, v_y, v_z).
  (We will usually use 2- or 3-dimensional vectors: e.g., **v** = (3, -2)).

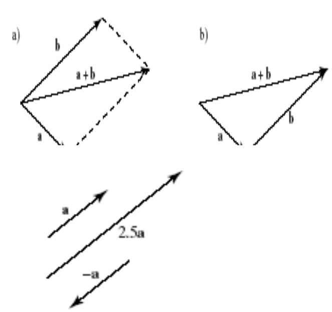
Vector Representations

- A vector **v** = (33, 142.7, 89.1) is a row vector.
- A vector **v** = $(33, 142.7, 89.1)^T$ is a column vector.
  – It is the same as

$$v = \begin{pmatrix} 33 \\ 142.7 \\ 89.1 \end{pmatrix}$$

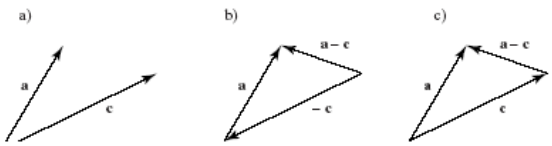
Vector Operations (2)

- Vectors have 2 fundamental operations: addition of 2 vectors and multiplication by a scalar.
- If **a** and **b** are vectors, so is **a** + **b**, and so is s**a**, where s is a scalar.



Vector Operations (3)

Subtracting **c** from **a** is equivalent to adding **a** and (-**c**), where –**c** = (-1)**c**.



Linear Combinations of Vectors

- $\mathbf{v}_1 \pm \mathbf{v}_2 = (v_{1x} \pm v_{2x}, v_{1y} \pm v_{2y}, v_{1z} \pm v_{2z})$
- $s\mathbf{v} = (sv_x, sv_y, sv_z)$
- A linear combination of the m vectors $\mathbf{v_1, v_2, \ldots, v_m}$ is $\mathbf{w} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_m\mathbf{v}_m$

Example:

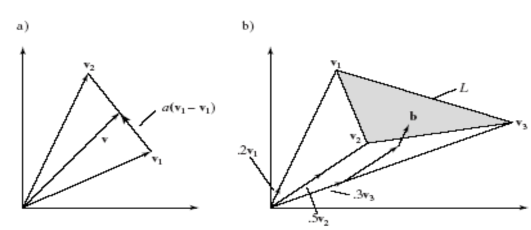
2(3, 4,-1) + 6(-1, 0, 2) forms the vector (0, 8, 10).

Linear Combinations of Vectors

- The linear combination becomes an affine combination if $a_1 + a_2 + \ldots + a_m = 1$.
  - Example: 3**a** + 2 **b** - 4 **c** is an affine combination of **a**, **b**, and **c**, but 3 **a** + **b** - 4 **c** is not.
  - (1-*t*) **a** + (*t*) **b** is an affine combination of **a** and **b**.
- The affine combination becomes a convex combination if $a_i \geq 0$ for $1 \leq i \leq m$.
  - Example: .3**a**+.7**b** is a convex combination of **a** and **b**, but 1.8**a** - 0.8**b** is not.

The Set of All Convex Combinations of 2 or 3 Vectors

- $\mathbf{v} = (1 - a)\mathbf{v}_1 + a\mathbf{v}_2$, as a varies from 0 to 1, gives the set of all convex combinations of $\mathbf{v}_1$ and $\mathbf{v}_2$. An example is shown below.



Vector Magnitude and Unit Vectors

- The magnitude (length, size) of *n*-vector **w** is written |**w**|. $|\mathbf{w}| = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$
- Example: the magnitude of **w** = (4, -2) is $\sqrt{20}$

  and that of **w** = (1, -3, 2) is $\sqrt{14}$
- A unit vector has magnitude |**v**| = 1.
- The unit vector pointing in the same direction as vector **a** is $\hat{a} = \frac{a}{|a|}$ (if |a| ≠0).
- Converting **a** to $\hat{a}$ is called *normalizing* vector **a**.

Vector Magnitude and Unit Vectors (2)

- At times we refer to a unit vector as a **direction**.
- Any vector can be written as its magnitude times its direction: **a** = |a| $\hat{a}$

Vector Dot Product

- The dot product of n-vectors **v** and **w** is

  $\mathbf{v}\cdot\mathbf{w} = v_1w_1 + v_2w_2 + \ldots + v_nw_n$
  - The dot product is commutative: **v**·**w** = **w**·**v**
  - The dot product is distributive: (**a** ± **b**)·**c** = **a**·**c** ± **b**·**c**
  - The dot product is associative over multiplication by a scalar: (s**a**)·**b** = s(**a**·**b**)
  - The dot product of a vector with itself is its magnitude squared: **b**·**b** = $|\mathbf{b}|^2$

Applications: Angle Between 2 Vectors

- **b** = (|**b**| cos ϕ_b, |**b**| sin ϕ_b), and

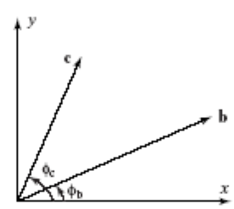
  c = (|**c**| cos ϕ_c, |**c**| sin ϕ_c)
- **b**·**c** = |**b**||**c**| cos ϕ_c cos ϕ_b + |**b**||**c**| sin ϕ_b sin ϕ_c

  = |**b**||**c**| cos (ϕ_c- ϕ_b)

  = |**b**||**c**| cos θ,

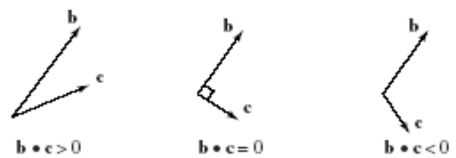
where θ = ϕ_c- ϕ_b is the smaller angle between **b** and **c**:

$$\cos(\theta) = \hat{\mathbf{b}} \cdot \hat{\mathbf{c}}$$



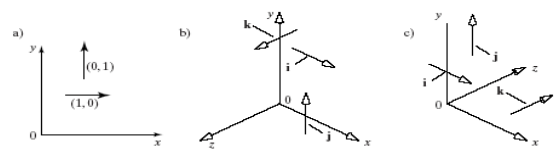
Angle Between 2 Vectors (2)

- The cosine is positive if $|\theta| < 90^o$, zero if $|\theta| = 90^o$, and negative if $\theta > 90^o$.
- Vectors **b** and **c** are perpendicular (orthogonal, normal) if **b**·**c** = 0.



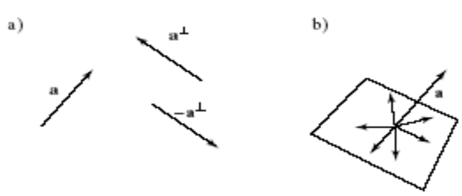
Standard Unit Vectors

- The standard unit vectors in 3D are **i** = (1,0,0), **j** = (0, 1, 0), and **k** = (0, 0, 1). **k** always points in the positive z direction
- In 2D, **i** = (1,0) and **j** = (0, 1).
- The unit vectors are orthogonal.



Finding a 2D "Perp" Vector

- If vector **a** = (a_x, a_y), then the vector perpendicular to **a** in the *counterclockwise* sense is $\mathbf{a}^{\perp} = (-a_y, a_x)$, and in the *clockwise* sense it is $-\mathbf{a}^{\perp} = (a_y, -a_x)$.
- In 3D, any vector in the plane perpendicular to **a** is a "perp" vector.



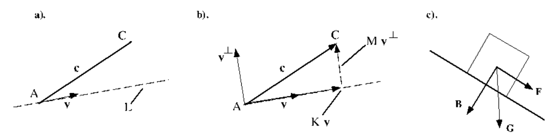
Properties of ⊥

- $(\mathbf{a} \pm \mathbf{b})^{\perp} = \mathbf{a}^{\perp} \pm \mathbf{b}^{\perp}$;
- $(s\mathbf{a})^{\perp} = s(\mathbf{a}^{\perp})$;
- $(\mathbf{a}^{\perp})^{\perp} = -\mathbf{a}$
- $\mathbf{a}^{\perp} \cdot \mathbf{b} = -\mathbf{b}^{\perp} \cdot \mathbf{a} = -a_y b_x + a_x b_y$;
- $\mathbf{a}^{\perp} \cdot \mathbf{a} = a \cdot \mathbf{a}^{\perp} = 0$;
- $|\mathbf{a}^{\perp}| = |\mathbf{a}|$;

Orthogonal Projections and Distance from a Line

- We are given 2 points A and C and a vector **v**. The following questions arise:
  - How far is C from the line L that passes through A in direction **v**?
  - If we drop a perpendicular onto L from C, where does it hit L?
  - How do we decompose a vector **c** = C – A into a part along L and a part perpendicular to L?

Illustration of Questions



Middle Tennessee State University

Answering the Questions

- We may write $\mathbf{c} = K\mathbf{v} + M\mathbf{v}^{\perp}$. If we take the dot product of each side with $\mathbf{v}$, we obtain $\mathbf{c}\cdot\mathbf{v} = K\mathbf{v}\cdot\mathbf{v} + M\mathbf{v}^{\perp}\cdot v = K|\mathbf{v}|^2$, or $K = \mathbf{c}\cdot\mathbf{v}/|\mathbf{v}|^2$.
- Likewise, taking the dot product with $\mathbf{v}^{\perp}$ gives $M = \mathbf{c}\cdot\mathbf{v}^{\perp}/|\mathbf{v}|^2$.
- Answers to the original questions: $M\mathbf{v}^{\perp}$, $K\mathbf{v}$, and both.

Middle Tennessee State University

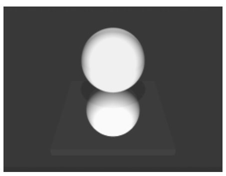
Practice Question

- Find the projection of the vector C=(6, 4) onto v=(1,2)
- How far is the point C=(6, 4) from the line that passes through (1, 1) and (4, 9)?

Middle Tennessee State University

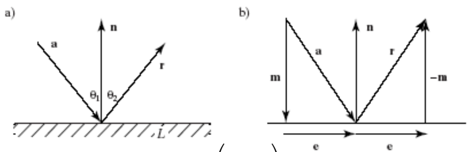
Application of Projection: Reflections

- A reflection occurs when light hits a shiny surface (below) or when a billiard ball hits the wall edge of a table.



Reflections (2)

- When light reflects from a mirror, the angle of reflection must equal the angle of incidence: $\theta_1 = \theta_2$.
- Vectors and projections allow us to compute the new direction **r**, in either two-dimensions or three dimensions.

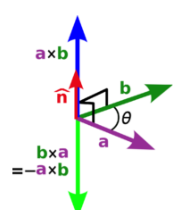


$$r = a - 2\left(\frac{a \cdot n}{|n|^2}\right)n$$
$$= a - 2(a \cdot \hat{n}) \cdot \hat{n}$$

Practice Question

- Given: a=(4, -2) and n=(0, 3)
  what is a's reflected light about n?

Cross Product



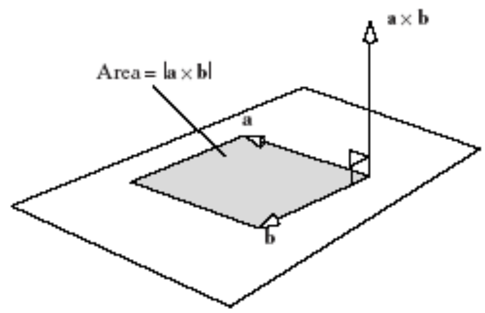
$$\mathbf{a} \times \mathbf{b} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}.$$

$$a \text{ x } b = i\begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} - j\begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} + k\begin{vmatrix} a_1 & a_2 \\ b_2 & b_2 \end{vmatrix}$$
$$= i(a_2b_3 - a_3b_2) - j(a_1b_3 - a_3b_1) + k(a_1b_2 - a_2b_2)$$

Properties of the Cross-Product

- **i** x **j** = **k**; **j** x **k** = **i**; **k** x **i** = **j**
- **a** x **b** = - **b** x **a**; **a** x (**b** ± **c**) = **a** x **b** ± **a** x **c**; (s**a**) x **b** = s(**a** x **b**)
- **a** x (**b** x **c**) ≠ (**a** x **b**) x **c**
  – for example, **a** = (a_x, a_y, 0), **b** = (b_x, b_y, 0), **c** = (0, 0, c_z)
- **c** = **a** x **b** is perpendicular to **a** and to **b**. The direction of **c** is given by a right/left hand rule in a right/left-handed coordinate system.

Geometric Interpretation of the Cross Product

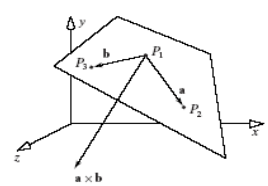


Properties (2)

- **a** · (**a** x **b**) = 0
- **|a** x **b|** = |**a**| |**b**| sin θ, where θ is the smaller angle between **a** and **b**.
- **|a** x **b|** is also the area of the parallelogram formed by **a** and **b**.
- **|a** x **b|** = 0 if **a** and **b** point in the same or opposite directions, or if one or both has length 0.

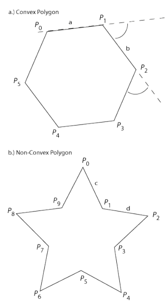
Application: Finding the Normal to a Plane

- Given any 3 non-collinear points A, B, and C in a plane, we can find a normal to the plane:
  - **a** = B – A, **b** = C – A, **n** = **a** x **b**. The normal on the other side of the plane is –**n**.



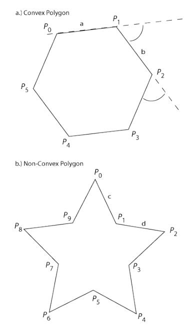
Convexity of Polygons

- Traversing around a <u>convex</u> polygon from one edge to the next, either a left turn or a right turn is taken, and they all must be the same kind of turn (all left or all right).
- An edge vector points along the edge of the polygon in the direction of travel.



Convexity of Polygons (2)

- Take the cross product of each edge vector with the next forward edge vector.
- If all the cross products point into (or all point out of) the plane, the polygon is convex; otherwise it is not.



Representations of Key Geometric Objects

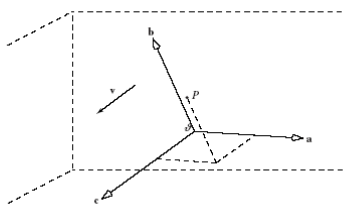
- Lines and planes are essential to graphics, and we must learn how to represent them – i.e., how to find an equation or function that distinguishes points on the line or plane from points off the line or plane.
- It turns out that this representation is easiest if we represent vectors and points using 4 coordinates rather than 3.

Coordinate Systems and Frames

- A vector or point has coordinates in an underlying coordinate system.
- In graphics, we may have multiple coordinate systems, with origins located anywhere in space.
- We define a coordinate frame as a single point (the origin, *O*) with 3 mutually perpendicular <u>unit</u> vectors: **a**, **b**, and **c**.

Coordinate Frames (2)

- A vector **v** is represented by (v_1, v_2, v_3) such that **v** = v_1**a** + v_2**b** + v_3**c**.
- A point P – *O* = p_1**a** +p_2**b** + p_3**c**.



Homogeneous Coordinates

- It is useful to represent both points and vectors by the same set of underlying objects, (**a**, **b**, **c**, *O*), *O* is the origin.
- A vector has no position, so we represent it as $$(a,b,c,O) \bullet \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix}$$
- A point has an origin (*O*), so we represent it by $$(a,b,c,O) \bullet \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix}$$

Changing to and from Homogeneous Coordinates

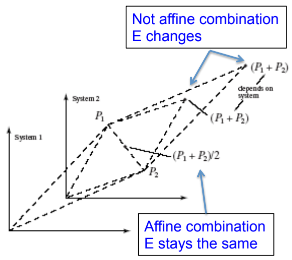
- To: if the object is a vector, add a 0 as the 4th coordinate; if it is a point, add a 1.
- From: simply remove the 4th coordinate.
- OpenGL uses 4D homogeneous coordinates for all its vertices.
  - If you send it a 3-tuple in the form (*x*, *y*, *z*), it converts it immediately to (*x*, *y*, *z*, 1).
  - If you send it a 2D point (*x*, *y*), it first appends a 0 for the z-component and then a 1, to form (*x*, *y*, 0, 1).
- All computations are done within OpenGL in 4D homogeneous coordinates.

Combinations

- Linear combinations of vectors and points:
  - The difference of 2 points is a vector: the fourth component is 1 – 1 = 0
  - The sum of a point and a vector is a point: the fourth component is 1 + 0 = 1
  - The sum of 2 vectors is a vector: 0 + 0 = 0
  - A vector multiplied by a scalar is still a vector: a x 0 = 0.
  - Linear combinations of vectors are vectors.

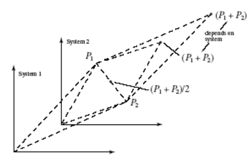
Combinations (2)

- The sum of 2 points: $E=a_1 \cdot P_1 + a_2 \cdot P_2$ is a point only if the points are part of an affine combination, so that $a_1 + a_2 = 1$. The sum is a vector only if $a_1 + a_2 = 0$.
- We require this rule to remedy the problem shown at right:



Combinations (3)

- If we form *any* linear combination of two points, say $E = fP + gR$, when $f + g$ is different from 1, a problem arises if we translate the origin of the coordinate system.
- Suppose the origin is translated by vector **u**, so that P is altered to P + **u** and R is translated to R + **u**.
- If E is a point, it must be translated to $E' = E$ + **u.**
- But we have $E' = fP + gR + (f + g)$**u,** which is *not* E + **u** unless $f + g = 1$.



Point + Vector

- Suppose we add a point A and a vector that has been scaled by a factor t: the result is a point,

  P = A + tv.

- Now suppose **v** = B – A, the difference of 2 points:

  P = tB + (1-t)A,

  an affine combination of points.

Linear Interpolation of 2 Points

- P = (1-t)A + tB is a linear interpolation (lerp) of 2 points. This is very useful in graphics in many applications,
  – P_x (t) provides an x value that is fraction t of the way between A_x and B_x. (Likewise P_y, P_z).

```
float lerp (float a, float b, float t)
{
    return a + (b – a) * t;  // return float
}
```

Middle Tennessee State University

Tweening and lerp

- One often wants to compute the point *P*(*t*) that is fraction *t* of the way along the straight line from point *A* to point *B* [the tween (for in-between) at *t* of points *A* and *B*].
- Each component of the resulting point is formed as the lerp () of the corresponding components of *A* and *B*.
- A procedure Tween (Point2 A, Point2 B, float t) implements tweening for points A and B, where we have used the new data type Point2 for a 2D point:

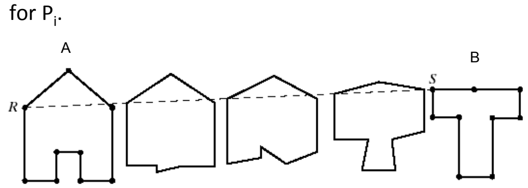
```
struct Point2
{
  float x;
  float y;
};
```


Tweening and Animation

- Tweening takes 2 polylines and interpolates between them (using lerp) to make one turn into another (or vice versa).
- We are finding here the point P(t) that is a fraction t of the way along the straight line from point A to point B.
- To start, it is easiest if one uses 2 polylines with the same number of lines.

Tweening (2)

- We use polylines A and B, each with n points numbered 0, 1, …, n-1.
- We form the points $P_i(t) = (1-t)A_i + tB_i$, for t = 0.0, 0.1, …, 1.0 (or any other set of t in [0, 1]), and draw the polyline for P_i.



Code for Tween

```
void drawTween(Point2 A[ ], Point2 B[ ], int n, float t)
{
  // draw the tween at time t between polylines A and B
  for (int i = 0; i < n; i++)
  {
      Point2 P;
      P = Tween (A[i], B[i], t);
      if (i ==0)
          moveTo(P.x, P.y);
      else
          lineTo(P.x, P.y);
  }
}
```

Tweening (3)

- To allow drawing tweens continuously, use the code below with double buffers.

```
for (t = 0.0, delT = 0.1; ; t += delT;)
{
   //clear the buffer
   drawTween (A, B, n, t);
   glutSwapBuffers();

   if ((t<=0.0) || (t>=1.0))
         delT = -delT;
}
```


Tween Examples

Uses of Tweening

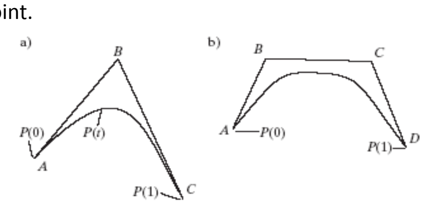
- In films, artists draw only the key frames of an animation sequence (usually the first and last).
  - Tweening is used to generate the in-between frames.
- Preview of Ch. 10: We want a smooth curve that passes through or near 3 points (A, B, and C). We expand $((1-t) + t)^2$ and write

$$P(t) = (1-t)^2A + 2t(1-t)B + t^2C$$

Uses of Tweening (2)

- This is called the Bezier curve for points A, B, and C.
- It can be extended to 4 points by expanding

$((1-t) + t)^3$ and using each term as the coefficient of a point.



Practice Questions

- What is the effect of tweening when all of the points A_i in polyline A are the same? How is polyline B distorted in its appearance in each tween?
- Polyline A is a square with vertices (1, 1), (-1, 1), (-1, -1), (1, -1) and polyline B is a wedge with vertices (4, 3), (5, -2), (4, 0), (3, -2). Sketch the shape P(t) for t=-1, -0.5, 0.5, and 1.5.