4. Vector Tools for Graphics

4.1 Basic types: scalars, points, vectors

- **Scalars:** real numbers (α, β, γ, ...)
- **Points:** locations in space (P, Q, R, ...)
- **Vectors:** quantities with direction and magnitude (u, v, w}, ...)

4.2 Vectors & their properties

\[ \mathbf{v} = \mathbf{Q} - \mathbf{P} \]
\[ P + \mathbf{v} = \mathbf{Q} \]
\[ \mathbf{u} + \mathbf{v} \]
\[ \alpha \mathbf{u} \]
\[ |\mathbf{u}| \quad (\text{norm}) \]
\[ \mathbf{u} \cdot \mathbf{v} = |\mathbf{n}| \quad |\mathbf{v}| \quad \cos(\theta) \]
\[ \mathbf{u} \times \mathbf{v} \]
\[ |\mathbf{u} \times \mathbf{v}| = |\mathbf{n}| \quad |\mathbf{v}| \quad \sin(\theta) \]
4.3 Homogeneous Coordinates

- ordinary representations of points and vectors are confusing and can make implementation difficult

**Coordinate Frame:** defined by \(( \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, P_0)\)

A point \(P\) is represented by

\[
P = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + P_0
\]

A vector \(\mathbf{v}\) is represented by

\[
\mathbf{v} = \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3
\]
homogeneous-coordinate representations:

\[
P : \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} \quad v : \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 0 \end{bmatrix}
\]

**Advantages:**

1. keep straight the distinction between points and vectors
2. provides compact notation when dealing with affine transformations

**Operations** valid with homogeneous coordinates?

- difference of two vectors
- sum of a point and a vector
- sum of two vectors
- vector scaling
- linear combination of vectors
Valid point operation: **Affine combination**

\[ \alpha_1 P_1 + \alpha_2 P_2 \cdots + \alpha_n P_n \]

with

\[ \alpha_1 + \alpha_2 \cdots + \alpha_n = 1 \]

**Why?**

Because affine combination is equivalent to a point plus a vector (a set of vectors)

**What is wrong with \( P + Q \)?**
- coordinate system dependent
4.4 Applications of Affine Combination

- Linear interpolation for "morphing"
- Linear interpolation for "inbetweening"
- Quadratic and cubic Bezier curves

**Representation of a Line:**

- Parametric form
- Point-normal form (for 2D only)

\[ L(t) = A + c \cdot t , \quad t \in R \]

\[ n \cdot (C - A) = 0 \]

(C: arbitrary point)
Representation of a Plane:

- Parametric form
- Point-normal form

\[ P(s, t) = B + a s + b t, \quad s, t \in R \]

\[ n \cdot (C - B) = 0 \quad (C: \text{arbitrary point}) \]
**Intersection of a Line with a Plane:**

\[
A \rightarrow t_{hit} \rightarrow A + n \cdot t
\]

\[n \cdot (C - B) = 0\]

\[t_{hit} = ?\]

**Intersection of two Planes:**

\[m \cdot (P - B) = 0\]

\[n \cdot (P - A) = 0\]

What is the parametric form of the intersecting line?
4.5 Polygon Intersection Problems

1. Is a given point $P$ inside or outside the object?

2. Where does a given ray $R$ first intersect the object?

3. Which part of a given line $L$ lies inside the object, and which part lies outside?

Looking for approaches that works for both 2D and 3D
Clipping line segment against convex polygon

- which part of $AC$ lies inside $P$?

Need:

- parametric representation of $AC$
- outward normal of each edge of $P$
Observations:

- $AC$ has two intersections with $P$

- $t_{in} = \max\{t_1, t_2, t_3\}$ \hspace{1cm} $t_{out} = \min\{t_4, t_5\}$

- $n \cdot c$ determines if we are entering or exiting the polygon $P$

- no need to check if an intersection lies "beyond" an endpoint
Cyrus-Beck Clipping Algorithm:

1. \( t_{in} \leftarrow 0; \)
   \( t_{out} \leftarrow 1; \)

2. For each edge \( e \)
   
   2.1. compute \( t_{hit} \);
      
      // parameter of the intersection point of the edge with \( AC \)

   2.2. if edge \( e \) is an entering edge,
      
      \( t_{in} \leftarrow \max \{ t_{hit}, t_{in} \} \);

      if edge \( e \) is an exiting edge,
      
      \( t_{out} \leftarrow \min \{ t_{hit}, t_{out} \} \);

   2.3. if \( t_{in} > t_{out} \), return; //nothing inside \( P \)

3. Output \( (A + c \ t_{in}) \) and \( (A + c \ t_{out}) \);
   
   // endpoints of the portion that is inside \( P \)
4.6 Polygon Clipping

- Can not simply use a line clipper since it may generate a series of unconnected line segments

A polygon clipper should generate one or more closed areas
Sutherland-Hodgman Algorithm

- clip polygon boundary against the four edges of the window separately

- For each edge of the window, traverse (directed) edges of the polygon and output vertices according to the following rules:

  - $v_2$ is output
    - (a) right edge
  - $i$ is output
    - (b) inside
  - No output
    - (c) outside
  - $i$ and $v_2$ are output
    - (d)
An Example
(clipping against the right edge of the window)

Start with $v_0v_1$
Output: $v_1$

Process $v_1v_2$
Output: $v_1v_1'$

Process $v_2v_3$
Output: $v_1v_1'v_2'v_3$

Process $v_3v_4$
Output: $v_1v_1'v_2'v_3v_3'$
Example (con’t)

Process $v_4v_5$
Output: $v_1v_1'v_2v_3v_3'$

Process $v_5v_0$
Output: $v_1v_1'v_2v_3v_3'v_5v_0$
Disadvantage of S-H algorithm:

- Output is always a connected area

Remedy: using Weiler-Atherton’s approach

For clockwise processing of polygon vertices in S-H clipping algorithm:

- For an outside-to-inside pair of vertices, follow the polygon boundary
- For an inside-to-outside pair of vertices, follow the window boundary in a clockwise direction
Can any of these algorithms be extended to a 3D algorithm?

Cyrus-Beck algorithm?

Sutherland-Hodgman algorithm?

Wiler-Atherton algorithm?