**Math Review**

### Products
- **Dot**
  \[
  \begin{bmatrix}
  x \\
  y \\
  z
  \end{bmatrix}
  \cdot
  \begin{bmatrix}
  a \\
  b \\
  c
  \end{bmatrix}
  = x \cdot a + y \cdot b + z \cdot c
  \]
  Useful for?
- **Cross**
  \[
  \begin{bmatrix}
  u_1 \\
  u_2 \\
  u_3
  \end{bmatrix}
  \times
  \begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3
  \end{bmatrix}
  =
  \begin{bmatrix}
  u_1v_3 - u_3v_1 \\
  u_2v_1 - u_1v_2 \\
  u_3v_2 - u_2v_3
  \end{bmatrix}
  \]
  Useful for?

---

### Coordinate Frame
- Coordinate frame: basis (independent) vectors + origin
  - can specify location - points

---

### Lines & Segments (2D)
- Segment \( \gamma_1 \) from \( P_i = (x_i, y_i) \) to \( P_i = (x_i, y_i) \)
  - Line through \( P_i = (x_i, y_i) \) and \( P_j = (x_j, y_j) \)
  - Parametric: \( G_i(t) = (x_i + t(x_j - x_i), y_i + t(y_j - y_i)) \), \( t \in [0,1] \)
  - Implicit: \( ax_i + by_i + c = 0 \)
  - Solve 2eq in 2 unknowns (e.g. set \( A = B = 1 \))
  - Explicit
  - 3D?

---

### Triangle
- **Normal**
  \[
  n = \frac{(P_2 - P_1) \times (P_2 - P_0)}{\| P_2 - P_1 \| \times \| P_2 - P_0 \|}
  \]
- **Area**
  \[
  A = \frac{1}{2} \left| \overrightarrow{P_0P_1} \times \overrightarrow{P_0P_2} \right|
  \]

---

**Copyright  2005 Alla Sheffer**
**Plane**

- Implicit equation: \( Ax + By + Cz + D = 0 \)
- Normalize (one option): \( A^2 + B^2 + C^2 = 1 \)
- \((A, B, C)\) - normal to plane

To find given 3 points \( P_0, P_1, P_2 \) in the plane:

\[
\mathbf{n} = (P_2 - P_0) \times (P_2 - P_1) / \|P_2 - P_0\| 
\]

Get \( n_x + n_y + n_z + D = 0 \) (solve 1 eq to get \( D \))

---

**Transformations Quiz**

- What do these transformations do?

\[
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & -1 & 0 \\
0 & -1 & 0 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 \\
1 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & a \\
0 & 1 \\
\end{bmatrix}
\]

---

**Transformations Quiz**

- And these homogeneous ones?

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & -1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0.5 \\
\end{bmatrix}
\]

---

**Transformations Quiz**

- How to mirror through arbitrary line in XY?

- What transformation achieves this?

---

**3D Transformations - Composition**

- Questions:
  - Is \( S_i S_j = S_j S_i \)?
  - Is \( T_i T_j = T_j T_i \)?
  - Is \( R_i R_j = R_j R_i \)?
  - Is \( S_i R_j = R_j S_i \)?
  - .....
Undoing Transformations: Inverses

\[ T(x,y,z)^{-1} = T(-x,-y,-z) \]
\[ T(x,y,z) \cdot T(-x,-y,-z) = I \]

\[ R(z,\theta)^{-1} = R(z,-\theta) = R^T(z,\theta) \quad \text{(R is orthogonal)} \]
\[ R(z,\theta) \cdot R(z,-\theta) = I \]

\[ S(x,y,z)^{-1} = S\left(\frac{1}{x},\frac{1}{y},\frac{1}{z}\right) \]
\[ S(x,y,z) \cdot S\left(\frac{1}{x},\frac{1}{y},\frac{1}{z}\right) = I \]

Another Transformations Quiz

- What does each transformation preserve?

<table>
<thead>
<tr>
<th>Transformation</th>
<th>lines</th>
<th>parallel lines</th>
<th>distance</th>
<th>angles</th>
<th>normals</th>
<th>convexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>scaling</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>rotation</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>translation</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>shear</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Composing Transformations

Suppose we want to rotate by 90 degrees and then translate by (2,3,0):

\[ P_a = \text{Rot}(z,-90)P_p \]
\[ P_b = \text{Trans}(2,3,0)P_a \]
\[ P_a = \text{Rot}(z,-90)P_p \]
\[ P_b = \text{Trans}(2,3,0)\text{Rot}(z,-90)P_p \]

Rotation About a Point: Moving Object

- \( P_{orig} = (x, y, 0) \)
- \( P'_{orig} = (x, y) \)
- \( P_{orig} = (x, y, z) \)
- \( P'_{orig} = (x, y, z) \)
- \( P_{orig} = (x, y, z) \)
- \( P'_{orig} = (x, y, z) \)

Composing Transformations

- \( R_{orig} = \text{Rot}(z,-90) \)
- \( R_{orig} = \text{Trans}(2,3,0) \)
- \( R_{orig} = \text{Shear}(sx, sy, sz) \)

Updates current transformation matrix by postmultiplying

\[ R_{trans} = \text{Rot}(z,-90) \cdot \text{Trans}(2,3,0) \cdot \text{Shear}(sx, sy, sz) \]
Transformation Hierarchies

Example

\[
glTranslatef(x, y, 0); 
glTranslatef(x, y, 0); 
glRotatef(\theta, 0, 0, 1); 
glRotatef(\theta, 0, 0, 1); 
glPushMatrix(); 
glPushMatrix(); 
\]

\[
\text{DrawBody();}
\text{DrawBody();}
\]

\[
\text{glPushMatrix();}
\text{glPushMatrix();}
\]

\[
\text{glTranslatef(0, 7, 0);}
\text{glTranslatef(0, 7, 0);}
\]

\[
\text{DrawHead();}
\text{DrawHead();}
\]

\[
\text{glPopMatrix();}
\text{glPopMatrix();}
\]

\[
\text{glPushMatrix();}
\text{glPushMatrix();}
\]

\[
\text{glTranslate(2.5, 5.5, 0);}
\text{glTranslate(2.5, 5.5, 0);}
\]

\[
\text{glRotatef(\theta, 0, 0, 1);}
\text{glRotatef(\theta, 0, 0, 1);}
\]

\[
\text{DrawUArm();}
\text{DrawUArm();}
\]

\[
\text{glTranslate(0, -3.5, 0);}
\text{glTranslate(0, -3.5, 0);}
\]

\[
\text{glRotatef(\theta, 0, 0, 1);}
\text{glRotatef(\theta, 0, 0, 1);}
\]

\[
\text{DrawLArm();}
\text{DrawLArm();}
\]

\[
\text{glPopMatrix();}
\text{glPopMatrix();}
\]

... (draw other arm)... (draw other arm)

Camera Description

- arbitrary viewing position
- eye point, look-at direction, up vector

\[
\text{WCS}
\]

\[
\text{eye}
\]

\[
\text{look-at}
\]

\[
P_{\text{eye}}
\]

\[
P_{\text{null}}
\]

\[
\text{gaze}
\]

Understanding Z

- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)

Perspective Projection

- Viewing is from point at finite distance
- Without loss of generality:
  - Viewpoint at origin
  - Viewing plane is z=d
- Given \(P=(x, y, z)\) triangle similarity gives:

\[
\frac{x}{z} = \frac{x_p}{d} \quad \text{and} \quad \frac{y}{z} = \frac{y_p}{d} \Rightarrow x_p = \frac{x}{z/d} \quad \text{and} \quad y_p = \frac{y}{z/d}
\]

OpenGL Perspective Derivation

Another Transformations Quiz

- What does each transformation preserve?
Midpoint (Bresenham) Algorithm

**Assumptions:**
- \(x_2 > x_1, y_2 > y_1\) and \(\frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1\)

**Idea:**
- Proceed along the line incrementally
- Have ONLY 2 choices
- Select one that minimizes error (distance to line)

Midpoint Line Drawing (cont’d)

- Starting point satisfies \(d(x_1,y_1) = 0\)
- Each step moves right (east) or upper right (northeast)
- Sign of \(d(x+1,y+\frac{1}{2})\) indicates if to move east or northeast

Bresenham Algorithm

**Distance (error):**
\[
s = \{ (x, y) \mid ax + by + c = xdy - ydx + c = 0 \}
\]
\[
d(x, y) = 2(xdy - ydx + c)
\]

**Assumptions:**
- Given point \(P = (x, y)\)
- \(d(x, y)\) is signed distance of \(P\) to \(s\) (up to normalization factor)
- \(d\) is zero for \(P \in s\)

Midpoint Line Algorithm (version 1)

- \[d(x, y) = (2ax + dy - 2c) + (2ay - 2dx + 2c) + 2(x + 1)(x + 1) + 5x = 0\]

Flood Fill Algorithm

**Input**
- polygon \(P\) with rasterized edges
- \(P = (x, y) \in P\) point inside \(P\)

Triangulation

- Convex polygons easily triangulated
- Concave polygons present a challenge
- Convexity - formal definition:
  \[S \text{ convex iff for any two points } P_i, Q_j \in S, tP_i + (1-t)Q_j \in S, t \in [0,1].\]
**Scanline Algorithm**

- Observation: Each intersection of straight line with boundary moves it from/into polygon
- Detect (set) pixels inside polygon boundary (simple closed curve) with set of horizontal lines (pixel apart)

```plaintext
for (y=yB; y<=yT; y++) {
    for (x=xL; x<=xR; x++)
        setPixel(x,y);
    xL += DxL;
    xR += DxR;
}
```

**Edge Walking**

```plaintext
for (y=yB; y<=yT; y++) {
    for (x=xL; x<=xR; x++)
        setPixel(x,y);
    xL += DxL;
    xR += DxR;
}
```

**Modern Rasterization**

- Define a triangle from implicit edge equations:

**Barycentric Coordinates**

- Area
  
  \[
  A = \frac{1}{2} \left| \mathbf{P}_1 \mathbf{P}_2 \times \mathbf{P}_3 \right|
  \]

- Barycentric coordinates
  
  \[
  a_i = A_{P_i P_1 P_2} / A, \quad a_2 = A_{P_1 P_2 P_3} / A, \quad a_3 = A_{P_1 P_2 P_3} / A,
  \]

- \[\mathbf{P} = a_1 \mathbf{P}_1 + a_2 \mathbf{P}_2 + a_3 \mathbf{P}_3\]

**Computing Barycentric Coords**

- Combining
  
  \[
  P = \frac{c_1}{c_1 + c_2} \mathbf{P}_1 + \frac{c_1}{c_1 + c_2} \mathbf{P}_2
  \]

- Gives
  
  \[
  P = \frac{c_1}{c_1 + c_2} \left( \frac{d_2}{d_1 + d_2} \mathbf{P}_1 + \frac{d_1}{d_1 + d_2} \mathbf{P}_2 \right) + \frac{c_1}{c_1 + c_2} \left( \frac{b_1}{b_1 + b_2} \mathbf{P}_1 + \frac{b_2}{b_1 + b_2} \mathbf{P}_2 \right)
  \]

**Cohen-Sutherland Algorithm (cont’d)**

- If \((x, y) \in \text{Window}\):
  
  ```plaintext
  \[
  C = \text{Code}(x, y) = (a_1, a_2, a_3, a_4, a_5,
  \]

- \(a_1 = x < \text{min}_x, \quad a_2 = x > \text{max}_x, \quad a_3 = y < \text{min}_y, \quad a_4 = y > \text{max}_y, \quad a_5 = x < 0, \quad a_6 = x > \text{width}, \quad a_7 = y < 0, \quad a_8 = y > \text{height})

- Code \((x, y)\):
  
  ```plaintext
  \[
  \text{Code}(x, y) = (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)
  \]

- \((x, y)\) is inside window if all code bits are 0.

Copyright 2005 Alla Sheffer
C-S Algorithm for convex polygons – full version

for i = 1 to n C_i = code(P_i)
if ((C_1 and C_2 and ... and C_n) = 0) then return;
if ((C_1 or C_2 or ... or C_n) = 0) then return poly;
else
for i = 1 to n if (OutsideWindow(P_i)) then begin
Edge = Window boundary of leftmost non-zero bit of C_i
P_{C_j} = \frac{P_j}{P_j} \text{ if no intersection return } P_{C_j}^*

P_{C_j} = \frac{P_j}{P_j} \text{ if no intersection return } P_{C_j}^*

if (C_{j-1} = P_{C_j}^* and C_{j+1} = P_{C_j}^*)
C-S-Clip(P_{C_j} = \frac{P_j}{P_j}, \ldots, P_{C_j}^*, \ldots, P_{C_j}^*, \ldots, P_{C_j}, \ldots, P_{n})
else if (P_{j-1} = P_{j}^*) (no intersection, or exactly on the end - vertex)
C-S-Clip(P_{j-1} = P_{j}^* \ldots, P_{j-1}^*, \ldots, P_{j+1}^*, \ldots, P_{n})
else if (P_{j+1} = P_{j}^*) (no intersection, or exactly on the end - vertex)
C-S-Clip(P_{j+1} = P_{j}^* \ldots, P_{j+1}^*, \ldots, P_{n})
end