University of British Columbia

Review 1

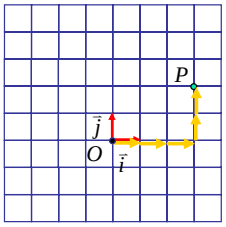
Products

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

University of British Columbia

Coordinate Frame

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

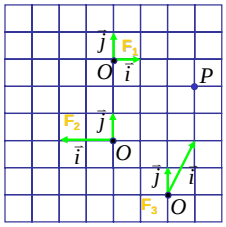


$P = O + x\vec{i} + y\vec{j}$

University of British Columbia

Math Review

- Working with Frames



$P = O + x\vec{i} + y\vec{j}$

F_1 P(3,-1)

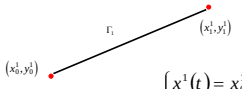
F_2 P(-1.5,2)

F_3 P(1,2)

University of British Columbia

Lines & Segments (2D)

- Segment Γ_1 from $P_0 = (x_0^1, y_0^1)$ to $P_1 = (x_1^1, y_1^1)$



$$G_1 = \begin{cases} x^1(t) = x_0^1 + (x_1^1 - x_0^1)t \\ y^1(t) = y_0^1 + (y_1^1 - y_0^1)t \end{cases} t \in [0,1]$$

- Line through $P_0 = (x_0^1, y_0^1)$ and $P_1 = (x_1^1, y_1^1)$
  - Parametric $G_1(t), t \in (-\infty, -\infty)$
  - Implicit $Ax+By+C=0$
    - Solve 2eq in 2 unknowns (e.g. set $A^2+B^2=1$)
  - Explicit
  - 3D?

University of British Columbia

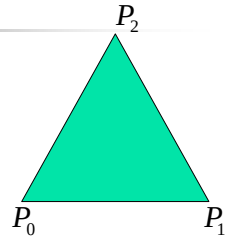
Triangle

- Normal

$$n = \frac{(P_1 - P_0) \times (P_2 - P_0)}{\|(P_1 - P_0) \times (P_2 - P_0)\|}$$

- Area

$$A = \frac{1}{2} \left\| \overrightarrow{P_0P_1} \times \overrightarrow{P_0P_2} \right\|$$



University of British Columbia

Plane

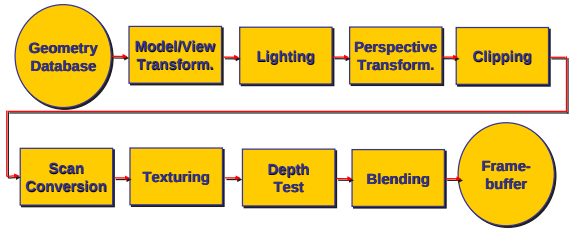
- Implicit equation: $Ax+By+Cz+D=0$
  - Normalize (one option): $A^2+B^2+C^2=1$
  - (A,B,C) - normal to plane
- To find given 3 points P_0 P_1 P_2 in the plane:

$$n = \frac{(P_1 - P_0)\times(P_2 - P_0)}{\|(P_1 - P_0)\times(P_2 - P_0)\|}$$

- Get $n_x x+n_y y+n_z z+D=0$ (solve 1 eq to get D)

University of British Columbia

The Rendering Pipeline



University of British Columbia

Transformations Quiz

- What do these transformations do?

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$

University of British Columbia

Transformations Quiz

- And these homogeneous ones?

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

University of British Columbia

Transformations Quiz

- How to mirror through arbitrary line in XY?
- What transformation achieves this?

University of British Columbia

3D Transformations - Composition

- Questions:
  - Is $S_1S_2 = S_2S_1$?
  - Is $T_1T_2 = T_2T_1$?
  - Is $R_1R_2 = R_2R_1$?
  - Is $S_1R_2 = R_2S_1$?
  - .....

University of British Columbia

Undoing Transformations: Inverses

$$\mathbf{T}(x,y,z)^{-1} = \mathbf{T}(-x,-y,-z)$$
$$\mathbf{T}(x,y,z)\,\mathbf{T}(-x,-y,-z) = \mathbf{I}$$

$$\mathbf{R}(z,\theta)^{-1} = \mathbf{R}(z,-\theta) = \mathbf{R}^{\mathrm{T}}(z,\theta)$$ **(R is orthogonal)**
$$\mathbf{R}(z,\theta)\,\mathbf{R}(z,-\theta) = \mathbf{I}$$

$$\mathbf{S}(sx,sy,sz)^{-1} = \mathbf{S}(\frac{1}{sx},\frac{1}{sy},\frac{1}{sz})$$
$$\mathbf{S}(sx,sy,sz)\,\mathbf{S}(\frac{1}{sx},\frac{1}{sy},\frac{1}{sz}) = \mathbf{I}$$

University of British Columbia

Another Transformations Quiz

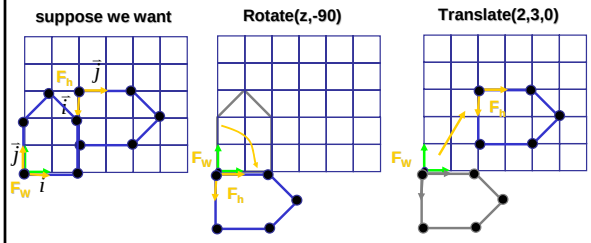
- What does each transformation preserve?

	lines	parallel lines	distance	angles	normals	convexity
scaling						
rotation						
translation						
shear						

University of British Columbia

Composing Transformations

suppose we want **Rotate(z,-90)** **Translate(2,3,0)**



$$P_A = Rot(z,-90)\,P_h \qquad P_W = Trans(2,3,0)\,P_A$$
$$P_W = Trans(2,3,0)Rot(z,-90)\,P_h$$

University of British Columbia

Composing Transformations

$$P_W = Trans(2,3,0)Rot(z,-90)\,P_h$$

- R-to-L: interpret operations wrt fixed coords
  - moving object
- L-to-R: interpret operations wrt local coords
  - changing coordinate system
- OpenGL (L-to-R, local coords)

```
glTranslatef(2,3,0);
glRotatef(-90,0,0,1);
DrawHouse();
```

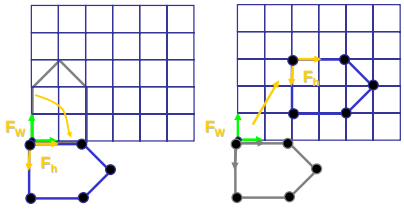
$$M_{MV} = Trans(2,3,0)\cdot M_{MV}$$
$$M_{MV} = Rot(z,-90)M_{MV}$$

updates current transformation matrix by postmultiplying

University of British Columbia

Composing Transformations

Rotate(z,-90) **Translate(-3,2,0) in local coords**



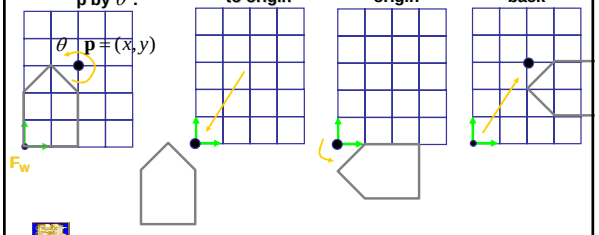
$$P_W = Rot(z,-90)Trans(-3,2,0)\,P_h$$

```
glRotatef(-90,0,0,1);
glTranslatef(-3,2,0);
draw_house();
```

University of British Columbia

Rotation About a Point: Moving Object

rotate about p by θ : **translate p to origin** **rotate about origin** **translate p back**

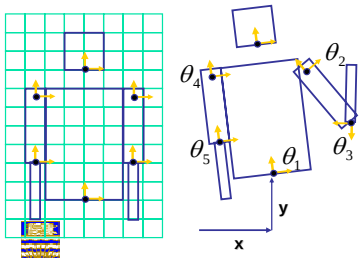


$$\mathbf{T}(x,y,z)\mathbf{R}(z,\theta)\mathbf{T}(-x,-y,-z)$$

University of British Columbia

Transformation Hierarchies

- Example

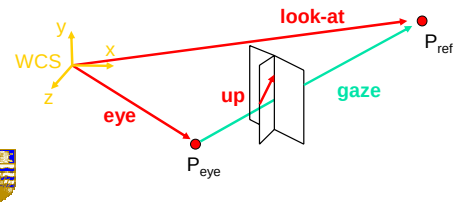



```
glTranslatef(x,y,0);
glRotatef(θ1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslatef(0,7,0);
  DrawHead();
glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef(θ2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef(θ3,0,0,1);
  DrawLArm();
glPopMatrix();
... (draw other arm)
```

University of British Columbia

Camera Description

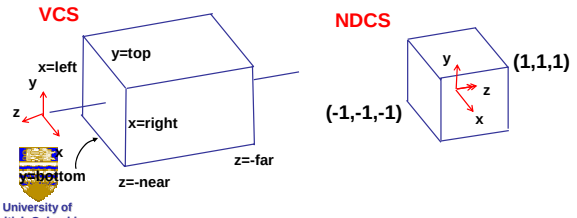
- arbitrary viewing position
  - eye point, look-at direction, up vector



University of British Columbia

Understanding Z

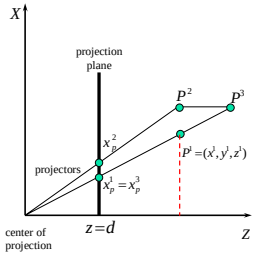
- z axis flip changes coord system handedness
  - RHS before projection (eye/view coords)
  - LHS after projection (clip, norm device coords)



University of British Columbia

Perspective Projection

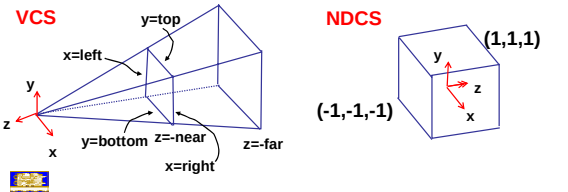
- Viewing is from *point at finite distance*
- Without loss of generality:
  - Viewpoint at origin
  - Viewing plane is $z=d$
- Given P=(x,y,z) triangle similarity gives:



$$\frac{x}{z}=\frac{x_p}{d} \text{ and } \frac{y}{z}=\frac{y_p}{d} \Rightarrow x_p=\frac{x}{z/d} \text{ and } y_p=\frac{y}{z/d}$$

University of British Columbia

OpenGL Perspective Derivation



University of British Columbia

Another Transformations Quiz

- What does each transformation preserve?

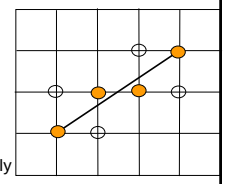
	lines	parallel lines	distance	angles	normals	convexity	conics
scaling							
rotation							
translation							
shear							
perspective							

University of British Columbia

Midpoint (Bresenham) Algorithm

- **Assumptions:**

$$x_2 > x_1, y_2 > y_1 \text{ and } \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1$$



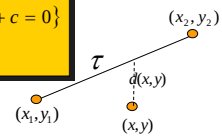
- **Idea:**
  - Proceed along the line incrementally
  - Have ONLY 2 choices
  - Select one that minimizes error (distance to line)

University of British Columbia

Bresenham Algorithm

Distance (error):

$$\tau = \{(x, y) ax + by + c = xdy - ydx + c = 0\}$$
$$d(x, y) = 2(xdy - ydx + c)$$

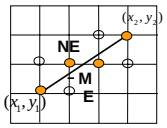


- Given point $P = (x, y)$, $d(x, y)$ is signed distance of p to τ (up to normalization factor)
- d is zero for $P \in \tau$

University of British Columbia

Midpoint Line Drawing (cont'd)

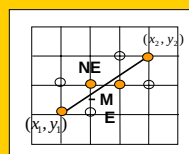
- Starting point satisfies $d(x_1, y_1) = 0$
- Each step moves right (east) or upper right (northeast)
- Sign of $d(x + 1, y + \frac{1}{2})$ indicates if to move east or northeast



University of British Columbia

Midpoint Line Algorithm (version 1)

```
Line ( x1, y1, x2, y2 )
begin
int  x, y, dx, dy, d ;
x ⇐ x1 ;          y ⇐ y1 ;
dx ⇐ x2 − x1 ;    dy ⇐ y2 − y1 ;
PlotPixel ( x, y ) ;
while ( x < x2 ) do
    d = (2x + 2)dy - (2y + 1)dx + 2c; // 2((x + 1)dy - (y + .5)dx + c)
    if ( d < 0 ) then
    begin
        x ⇐ x + 1 ;
    end ;
    else begin
        x ⇐ x + 1 ;
        y ⇐ y + 1 ;
    end ;
    PlotPixel ( x, y ) ;
end ;
end ;
```



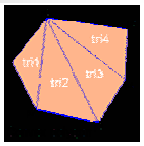

bresenham

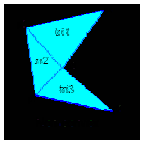
University of British Columbia

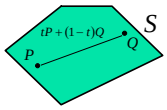
Triangulation

- Convex polygons easily triangulated
- Concave polygons present a challenge
- Convexity - formal definition:

Object S is **convex** iff for any two points $P, Q \in S$, $tP + (1-t)Q \subseteq S$, $t \in [0,1]$.



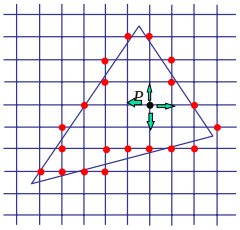




University of British Columbia

Flood Fill Algorithm

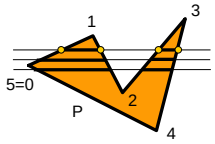
- Input
  - polygon P with rasterized edges
  - $P = (x,y) \in P$ point inside P



University of British Columbia

Scanline Algorithm

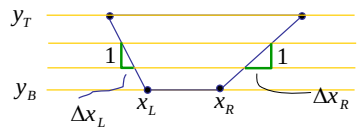
- Observation: Each intersection of straight line with boundary moves it from/into polygon
- Detect (& set) pixels inside polygon boundary (simple closed curve) with set of horizontal lines (pixel apart)



University of British Columbia

Edge Walking

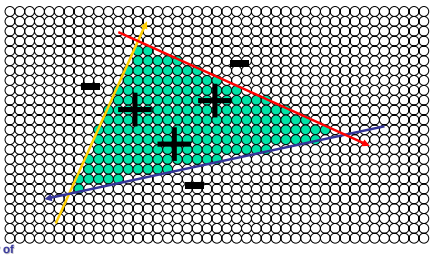
```
for (y=yB; y<=yT; y++) {
  for (x=xL; x<=xR; x++)
    setPixel(x,y);
  xL += DxL;
  xR += DxR;
}
```



University of British Columbia

Modern Rasterization

- Define a triangle from implicit edge equations:



University of British Columbia

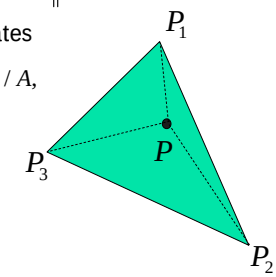
Barycentric Coordinates

- Area

$$A = \frac{1}{2}\left\|\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}\right\|$$

- Barycentric coordinates

$$a_1 = A_{P_2P_3P} / A,\ a_2 = A_{P_3P_1P} / A,$$
$$a_3 = A_{P_1P_2P} / A,$$
$$P = a_1P_1 + a_2P_2 + a_3P_3$$



University of British Columbia

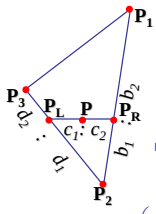
Computing Barycentric Coords

- combining

$$P = \frac{c_2}{c_1+c_2}\cdot P_L + \frac{c_1}{c_1+c_2}\cdot P_R$$

$$P_L = \frac{d_2}{d_1+d_2}P_2 + \frac{d_1}{d_1+d_2}P_3$$

$$P_R = \frac{b_2}{b_1+b_2}P_2 + \frac{b_1}{b_1+b_2}P_1$$



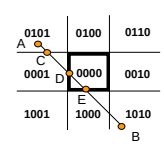
- gives

$$P = \frac{c_2}{c_1+c_2}\left(\frac{d_2}{d_1+d_2}P_2 + \frac{d_1}{d_1+d_2}P_3\right) + \frac{c_1}{c_1+c_2}\left(\frac{b_2}{b_1+b_2}P_2 + \frac{b_1}{b_1+b_2}P_1\right)$$

University of British Columbia

Cohen-Sutherland Algorithm (cont'd)

C-S-Clip($P_0 = (x_0, y_0), P_1 = (x_1, y_1), x_{min}, x_{max}, y_{min}, y_{max}$)
$C_0 \Leftarrow$ code (P_0); $C_1 \Leftarrow$ code (P_1);
if ((C_0 and C_1) != 0) then return;
if ((C_0 or C_1) == 0) then draw(P_0, P_1);
else if (OutsideWindow(P_0)) then
begin
Edge $\Leftarrow$ Window boundary of leftmost non-zero bit of C_0;
$P_2 \Leftarrow \overline{P_0, P_1} \cap Edge$;
C-S-Clip($P_2, P_1, x_{min}, x_{max}, y_{min}, y_{max}$);
end
else
Edge $\Leftarrow$ Window boundary of leftmost non-zero bit of C_1;
$P_2 \Leftarrow \overline{P_0, P_1} \cap Edge$;
C-S-Clip($P_0, P_2, x_{min}, x_{max}, y_{min}, y_{max}$);
end



AB → CB → DB → DE

bit	1	0
1	$y < y_{min}$	$y \geq y_{min}$
2	$y > y_{max}$	$y \leq y_{max}$
3	$x > x_{max}$	$x \leq x_{max}$
4	$x < x_{min}$	$x \geq x_{min}$

University of British Columbia

C-S Algorithm for convex polygons – full version

```
C-S-Clip( poly = P_0,..., P_n, x_min, x_max, y_min, y_max )
for i = 1 to n C_i ⇐ code (P_i);
if ((C_0 and C_1 and ... and C_n) != 0) then return;
if ((C_0 or C_1 or ... or C_n) == 0) then draw( 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_{i-1,i} ⇐ P_{i-1}, P_i ∩ Edge; /* if no intersection return P_{i-1} */
  P_{i,i+1} ⇐ P_i, P_{i+1} ∩ Edge; /* if no intersection return P_{i+1} */
  if (P_{i-1,i} == P_{i-1} and P_{i,i+1} == P_{i+1})
     C-S-Clip( P_0,..., P_{i-1}, P_{i+1},..., P_n, x_min, x_max, y_min, y_max )
  else if (P_{i-1,i} == P_{i-1}) /* no intersection, or exactly on the end-vertex */
     C-S-Clip( P_0,..., P_{i-1}, P_{i,i+1}, P_{i+1},..., P_n, x_min, x_max, y_min, y_max );
  else if (P_{i,i+1} == P_{i+1}) /* no intersection, or exactly on the end-vertex */
     C-S-Clip( P_0,..., P_{i-1}, P_{i-1,i}, P_{i+1},..., P_n, x_min, x_max, y_min, y_max )
  else
     C-S-Clip( P_0,..., P_{i-1}, P_{i-1,i}, P_{i,i+1}, P_{i+1},..., P_n, x_min, x_max, y_min, y_max );
end
```

bit	1	0
1	$y<y_{min}$	$y \geq y_{min}$
2	$y>y_{max}$	$y \leq y_{max}$
3	$x>x_{max}$	$x \leq x_{max}$
4	$x<x_{min}$	$x \geq x_{min}$