1. glDepthFunc(): specify value used for depth buffer comparison
   → GL_LEQUAL: Passes if the incoming depth value is <= to the stored depth value
   {

2. glEnable(): Enable server-side GL capabilities.
   → GL_DEPTH_TEST: If enable, do depth comparison and update buffer
   → GL_CULL_FACE: If enable, cull polygons based on their winding in window coordinates

3. glViewport: set viewport
   ↳ (x, y, width, height)
   - x, y: lower left corner of the viewport rectangle (initial is 0,0)
   - glViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates.

$$x_w = x_{nd} + 1 \frac{width}{2} + x$$
$$y_w = y_{nd} + 1 \frac{height}{2} + y$$

~~4. glMatrixMode~~
4. glMatrixMode(mode): specify which matrix is the current matrix.
   specifies which matrix is the target for subsequent matrix operations
   - GL_PROJECTION: Applies subsequent matrix operation to the projection matrix stack
   - GL_MODELVIEW: " " " " " modelview " "

5. glLoadIdentity: replace the current matrix w/ the identity matrix
   equivalent to glLoadMatrix of identity matrix but more efficient

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

6. gluOrtho2D(left, right, bottom, top) defines a 2D orthographic projection matrix.
   → bottom, top: horizontal clipping planes
   → left and vertical clipping planes

7. gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
   ↳ define viewing transformation
   (eye: eye; reference: center; up vector: up)

~~gl Pop matrix (Pop current matrix stack~~

8 gl Push Matrix pushes current matrix stack down by one, duplicating the matrix

gl Pop matrix pops the current matrix stack, replaces current matrix w/ one below it on the stack

9. glu Perspective (fovy, // field of view angle in the y direction
xspect, // xspect ratio that determine the field of view in x direction
zNear, // Distance from viewer to near clipping plane (always positive)
zFar); // distance the viewer to the far clipping plane (always positive)

top_left = 0, top_right, bottom_right, bottom_left

glu Sphere: quad, radius, slices, stacks
- quad: quadrics objects
- radius of object
- # of subdivision around z-axis (longitude)
- # of subdivision along z-axis (latitude)

glu cylinder:

quad
base radius
top radius
height
slices
stack

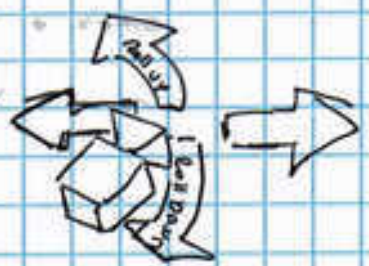
ordinary order

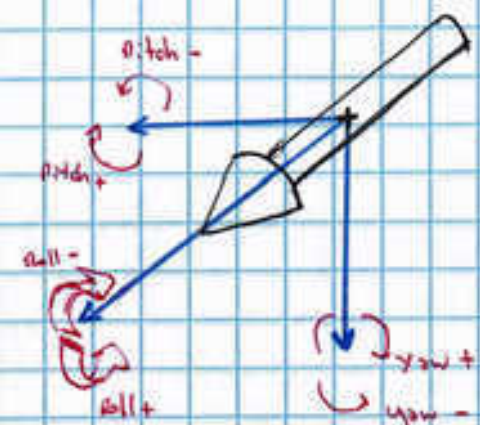
Front view: camera x = 0.0f ref x = 0.0f top: x = 0.0f
y = 0.0f y = 0.0f y = 0.0f
z = variable z = 0.0f z = 0.0f

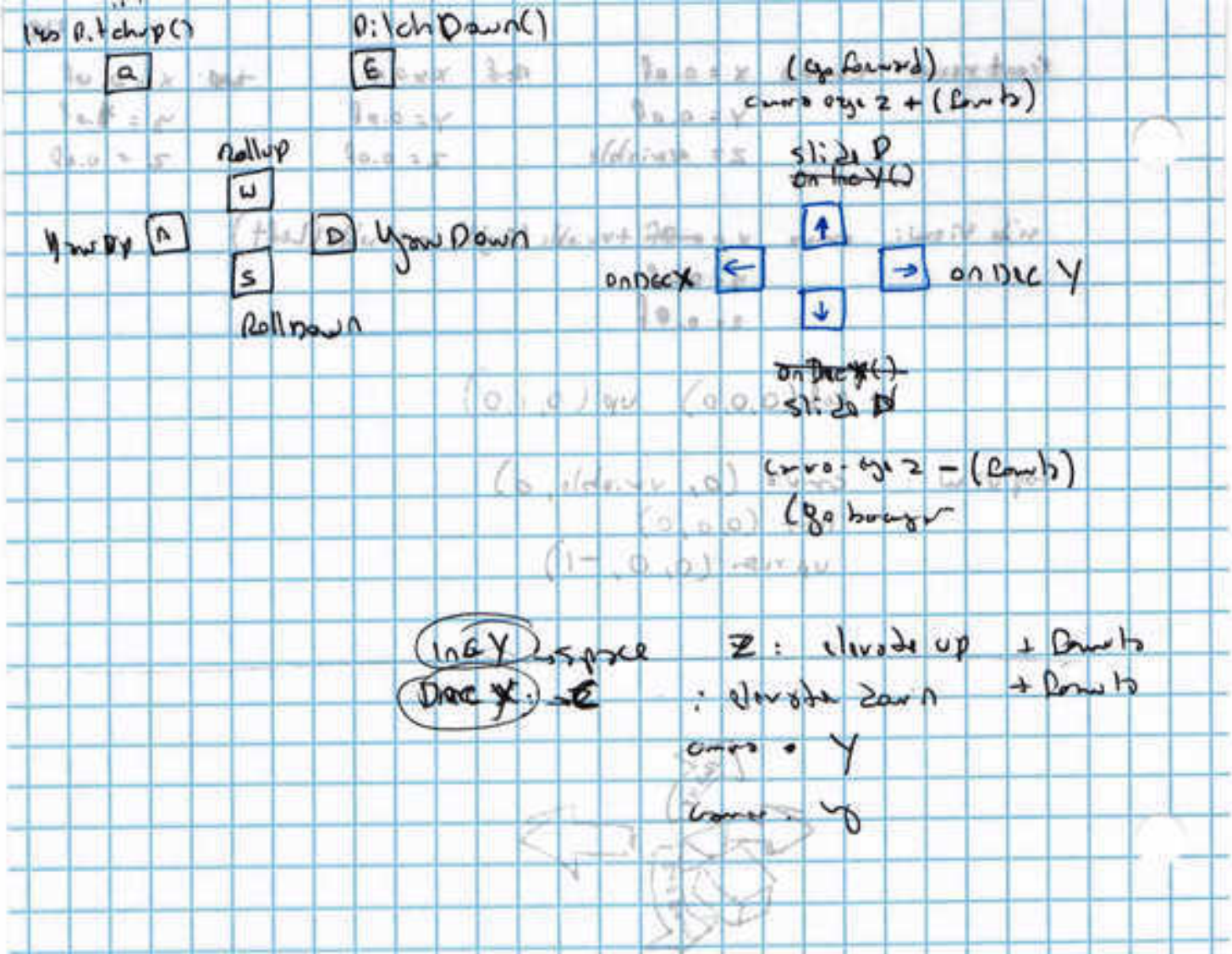
side view: cam x = ~~0.0f~~ +variable (Right) -variable (Left)
y = 0.0f
z = 0.0f

ref (0,0,0) up (0,1,0)

top view: cam (0, variable, 0)
ref (0,0,0)
up vector (0,0,-1)






Pitch Up()
Q
Pitch Down()
E
(Go Forward)
curr.pos.z + (Bounds)
Roll Up
W
Yaw By
A
D
Yaw Down
S
Roll Down
Slide P
on Dec Y()
on Dec X
on Dec Y
Slide D
curr.pos.z - (Bounds)
(Go Backwards
InC Y : space
Dec X : Z
Z : elevate up + Bounds
: elevate down + Bounds
curr . Y
curr . y

```
gluLookAt(camera.x,
          .y,
          .z,
          n.x,
          n.y,
          n.z,
          v.x),
          v.y)
          v.z)
```

s1, 0.5, 0.5, 20.0, 10, 20

s2, 0.5, 0.5, 10, 10, 20

s3, 0.5 0.5, 10, 10, 20

s4, 0.0, 0.5, 10, 10, 20

s5, 0, 0, 0.5, 10. 10, 20

(b1, 1.0, 10.0, 10.0)

b2, 1.0, 10.0, 10.0

b3 " "

(b4 2.0, 10, 10)

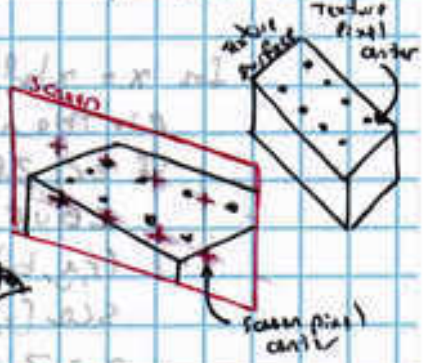
(b5 2.0, 10, 10)

Image warping: Act of distorting a source image into a destination image

mapping

source space $(u, v) \rightarrow$ destination image (x, y)

1. Mapping functions:
   a. $x(u, v)$
   b. $y(u, v)$
2. ~~"warp" = distortion & "translation"~~
   - warp = signal → mapping of the domain of an image
   - translation = mapping of the image range values
3. When rendering a texture surface it is necessary to sample the texture image to produce the screen image
   a. Except for scenes viewing w/ parallel projections, texturing mappings are non-affine, so they have non uniform resampling grids.
   b. Affine maps are those composed of rotations, scales, and translations.



(Textured surface viewed in perspective)

4. Texture space → 2D space of surface textured
   Object space → 3D geometry (Polygons, ect)
5. Texture coordinate (u, v) are assigned to each vertex
6. 2D texture space
   ↓ [Parameterization]
   3D - Object space
   ↓ [Projection]
   2D screen space

   (compound mapping)

7. Naive implementation of texture mapping

a. texture mapping w/in a z-buffer polygon renderer.

b. i. define object space coordinates (x,y,z)

ii. define texture coordinates (u,v) at each vertex of each polygon

iii. transform polygon to screen space, giving x, y, and z coordinates for each vertex.

iv. compute z, u, and v at each pixel.

v. linear interpolation is used for computing z.

vi. scan polygon and convert polygon into pixels by linear interpolation of x, z, u, and v, along left and right sides of the polygon and linearly interpolating z, u, and v across each scan line.

vii. Have texture coordinate (u,v) at each pixel, map

I. sample the texture array

II. use the result to color the screen pixel value.

viii Inner loop:

```
for x = xleft to x right
    // is the new point closer?
    if z < ZBUF[x,y] then
        ZBUF[x,y] = z        // update z-buffer
        (r,g,b) = TEX[u,v]   // sample texture
        SCR[x,y] = (r,g,b)   // write color
        z = z + dz
        (u,v) = (u,v) + (du, dv)
```

I. dz, du, dv are calculated once per scan line

Inferring Projective Mapping

1. specify four corners of source & destination quadrilaterals with a mouse to warp one area to the other.

2. Projective mapping w/ 8 degrees of freedom:
   a. correspondence map (u_k, v_k) to (x_k, y_k) for vertices $k = 0, 1, 2, 3$...
   b. Compute forward mapping matrix H and
      I. $i = 1$, switch unknowns $x \to y$
      II. $$x_k = \frac{a u_k + b v_k + c}{g u_k + h v_k + 1}$$

$$= u_k a + v_k b + c - u_k x_k g - v_k x_k h$$

for $k = 0, 1, 2, 3$.

$$y_k = \frac{d u_k + e v_k + f}{g v_k + h v_k + 1}$$

$$= u_k d + v_k e + f - u_k y_k g - v_k y_k h$$

for $k = 0, 1, 2, 3$

   c. rewritten as a matrix:

$$\begin{bmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0y_0 & -v_0y_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2y_2 & -v_2y_2 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3y_3 & -v_3y_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

3. Solve:

a. case 1: solve symbolically where u or v or quadrilateral is a unit square.

I. vertex correspondences:

x	y	u	v
x_0	y_0	0	0
x_1	y_1	1	0
x_2	y_2	1	1
x_3	y_3	0	1

II. $c = x_0$

$$a + c - g x_1 = x_1$$

$$a + b + c - g x_2 - h x_2 = x_2$$

$$b + c - h x_3 = x_3$$

$$f = x_0$$

$$d + f - g y_1 = y_1$$

$$d + e + f - g y_2 + h y_2 = y_2$$

$$e + f - h y_3 = y_3$$

III. Define:

- $\Delta x_1 = x_1 - x_2$
- $\Delta y_1 = y_1 - y_2$
- $\Delta x_2 = x_3 - x_2$
- $\Delta y_2 = y_3 - y_2$
- $\Sigma x = x_0 - x_1 + x_2 - x_3$
- $\Sigma y = y_0 - y_1 + y_2 - y_3$

IV two subcases:

A. If $\Sigma x = 0$ & $\Sigma y = 0$

- ~~xy plane~~ xy polygon is parallelogram so region is affine
- $a = x_1 - x_0$
  $b = x_2 - x_1$
  $c = x_0$
  $d = y_1 - y_0$
  $e = y_2 - y_1$
  $f = y_0$
  $g = 0$
  $h = 0$

B. If $\Sigma x \neq 0$ || $\Sigma y \neq 0$

- Reflection region =

- $g = \dfrac{\begin{vmatrix} \Sigma x & \Delta x_2 \\ \Sigma y & \Delta y_2 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}$

- $h = \dfrac{\begin{vmatrix} \Delta x_1 & \Sigma x \\ \Delta y_1 & \Sigma y \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}$

- $a = x_1 - x_0 + gx_1$
- $b = x_3 - x_0 + hx_3$
- $c = x_0$
- $d = y_1 - y_0 + gy_1$
- $e = y_3 - y_0 + hy_3$
- $f = y_0$

Mapping to Polygon Mesh Object

- Mapping a square to a quadrilateral. □ ↔ ⬠

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} s' \\ t' \\ q \end{bmatrix} \qquad (x, y) = \left(\frac{x'}{w}, \frac{y'}{w} \right)$$

$$(s, t) = \left(\frac{s'}{q}, \frac{t'}{q} \right)$$

- Affine (linear) Mapping:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \qquad \begin{aligned} x &= as + bt + c \\ y &= ds + et + f \end{aligned}$$

- Forward Mapping

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}$$

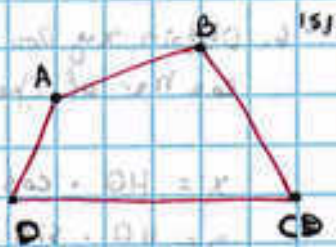
- Inverse Mapping

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{\text{cofactor}(Tf)}{\det(Tf)} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \frac{1}{ae - bd} \cdot \begin{bmatrix} e & -b & bf - ec \\ -d & a & dc - af \\ 0 & 0 & ae - bd \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

150

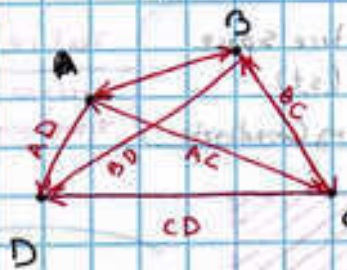
1. Let's ABCD be a quadrilateral where:
   A is the top-~~right~~ left in which the pair (xA, yA) represent the x y coordinates of this vertex
   B is the top-Right with (xB, yB)
   C is the bottom-Right with (xC, yC)
   D is the bottom-left with (xD, yD)



2. We are mapping points in this quadrilateral to the square EFGH whose side has length equal to m value

follows

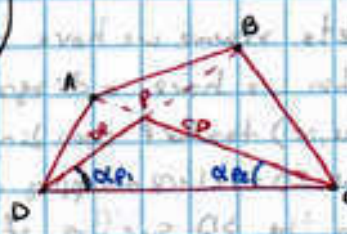
3. We need to compute the lengths:
   $AD = ((xA - xD)^2 + (yA - yD)^2)^{1/2}$
   $CD = ((xC - xD)^2 + (yC - yD)^2)^{1/2}$
   $AC = ((xA - xC)^2 + (yA - yC)^2)^{1/2}$
   $BD = ((xB - xD)^2 + (yB - yD)^2)^{1/2}$
   $BC = ((xB - xC)^2 + (yB - yC)^2)^{1/2}$

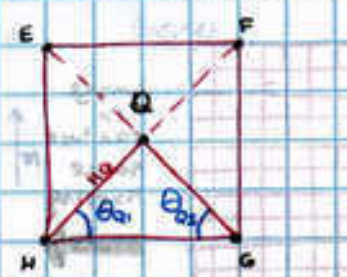


4. Next, we need to compute αD and αC.
   a. αD (at the vertex D) $= \arccos\left(\frac{(AD^2 + CD^2 - AC^2)}{2 \cdot AD \cdot CD}\right)$

   b. αC (at vertex C) $= \arccos\left(\frac{(BC^2 + CD^2 - BD^2)}{2 \cdot BC \cdot CD}\right)$

5. Map each point P in the quadrilateral to a point Q in the square
   $DP = ((xP - xD)^2 + (yP - yD)^2)^{1/2}$
   $CP = ((xP - xC)^2 + (yP - yC)^2)^{1/2}$
   $\alpha P_1 = \arccos\left(\frac{(DP^2 + CD^2 - CP^2)}{2 \cdot DP \cdot CD}\right)$
   $\alpha P_2 = \arccos\left(\frac{(CP^2 + CD^2 - DP^2)}{2 \cdot CP \cdot CD}\right)$





$\theta Q_1 = \frac{\alpha P_1 \cdot 90}{\alpha D}$

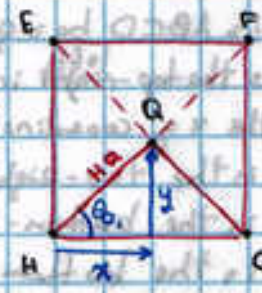
$\theta Q_2 = \frac{\alpha P_2 \cdot 90}{\alpha C}$

$HQ = \frac{m \cdot \sin(\theta Q_2)}{\sin(180 - \theta Q_1 - \theta Q_2)}$

6. Obtain xy position of Q relative to the Bottom-left corner of the square EFGH:

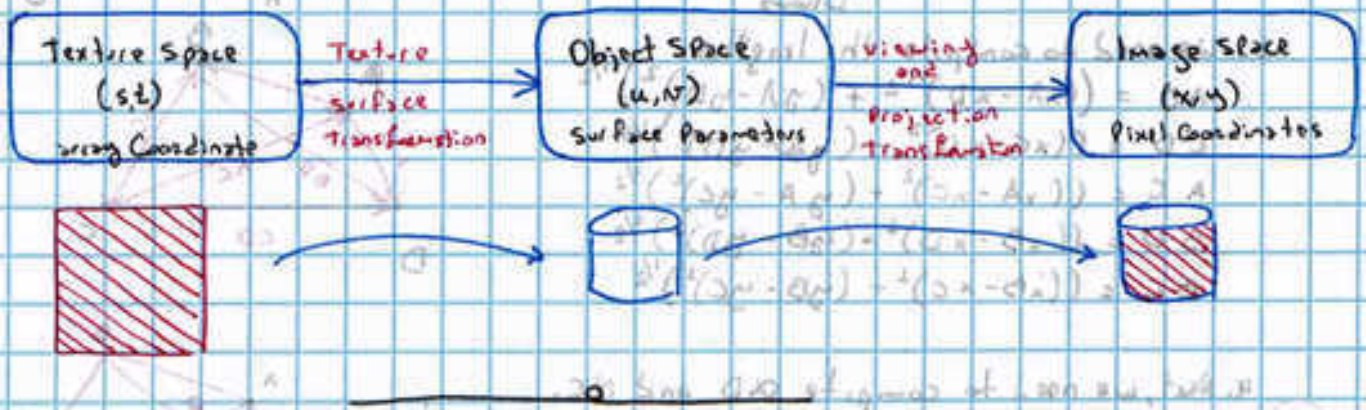
$x = HQ \cdot \cos(\theta_{01})$

$y = HQ \cdot \sin(\theta_{02})$



7. Keep track of the mapped color values of each point in the square so the average color in each of those points can be calculated

---



2D texture mapping consist of mapping a flat 2D bitmap image onto a surface (flat or curved)

1. Lets assume we have a 512×512 bitmap image
2. Then we have a square consisting 3 polygons which we assign (u,v) texture coordinate at their corners
3. The 2D texture mapping process is done by mapping the bitmap pixels to the 3D surface of the polygons