Computer Graphics . Review . Final needs 90%+ to get A .

Preface
Java & OpenGL .

Ch 1. Introduction

1. Computer Graphics :
   refer to anything involved in the creation / manipulation of images on computer, including animated images .

1.1 Painting & Drawing

1. An image presented on a computer screen is made up of pixels .
   ① At any given time,
      each pixel can show only one color .
   ② 24-bit color :
      RGB . 3 8-bit no .
   ③ Grayscale :
      black-to-white scale . 256 shades .

2. Indexed Color Display :
   a numbered list of possible colors &
   the color of a pixel is specified by an int giving the position of the color in list .

3. Frame Buffer :
   the color values for all the pixels on the screen are stored in a large block of memory → frame buffer . (temp. storage) ,
   changing vals in frame buffer
   → changes images on the screen .

* pixel coor. sys. is machine dependent .

lossless data compression .

lossy

4 . Raster Graphics . eg . GIF , PNG , JPEG .

an image consisting of a grid of pixels w/ numerical color values for each pixel .

① electron beam :

move along rows of pixels & glow . (RGB) .

② numerical val :

light intensity

⊕ photographic image

⊖ more space required .

=> Modern Raster Graphics

✓ screen made up of pixels .

✓ color vals for all pixels stored in frame buffer .

× no electron beam .

✓ update image via updating frame buffer vals .

5 . Vector Graphics : eg . SVG . objects .

an image as a list of geometric shapes (lines, triangles, etc.) that it contains .

① Shapes have attributes .

eg . thickness , color ,

② electron beam :

directly draw a line on screen & sweep along it .

③ vector graphics display :

store a display list of lines that should appear on screen . & go through the list over & over .

⊕ less space needed to store info .

⊕ great for blueprints & illustrations

⊖ not photographic .

=> ✓ change image via changing display list .

✓ only need to store coordinates of major vertices .

✓ faster than raster graphics (frame buffer → screen)

6. Painting programs : → Raster
image as a grid of pixels.
user create an image by assigning colors to pixels.
* only the pixel colors are saved
⊖ disappearing overlayed images.
(one pixel holds only one color at a time).

7. Drawing Prog : → Vector.
Create an image by adding geometric shapes, &
image represented as a list of these shapes.
⊕ preserve overlapping shapes
⊕ rich editing options:
translate, scale, ...

8. Coordinate System:
set up a correspondence b/t numbers & geometric pts.
① Raster Image:
2D grid of pixels in rows & cols.
(int vals)
② Vector Image:
real-no. coor.

$\boxed{\text{1.2 Elements of 3D Graphics}}$

1. Geometric Modeling:
use a list of geometric objects to represent an image.

2. Projection:
equiv. to taking a photograph of the scene.
2D projection of a 3D ~~image~~ scene.

3. World Coordinates:
x, y, z

4. Geometric Primitives: (sys. dependent).
the smallest building blocks
eg. line segments, triangles.

5. Hierarchical Modeling:

use already designed geometric model as a component in more complex models.

6. Geometric Transform :

used to adjust size, orientation, position of a geometric object.

① Scaling :

set size of object by some factor

② Rotation :

set orientation by rotating it by some angle about some specific axis

③ Translation :

set position by displacing it by a given amount from it original position.

7. Material :

how the surface interacts w/ light

① Shininess

② Roughness

③ Transparency.

texture

→ depend on lighting.

each light source has its own

color

intensity

direction / position.

8. Rasterization :

assign colors to individual pixels in the 2D image

& ⇒ 3D.

* the whole process of producing an image is "rendering the scene".

Ch2 2D Graphics

2.1 Pixels, Coordinates, Colors

\# Coord. Sys. associate no. to pts
\# color Model associate no. to colors.

2.1.1 Pixel Coor.

1. A pixel : identified by 2 int (row & col #).
   eg. (3,5)
   ↳ col no. 3 & row no. 5
   rows : no. bottom up v. top down
   cols : no. left to right.
   * sys. dependent.
   ⊖ pixels are approx.

2. Aliasing :
   ideal images w/ real-no. coord. will map several pts to the same pt. of int pixel coord.
   ⇒ jagged staircase line

3. Anti-Aliasing :
   when a pixel is only partially covered by a shape, the color of the pixel should be a mixture of the color of the shape & color of background,
   ⊕ reduce jaggies.

* pixel coord.
refer to the top-left corner of the pixel. (lines of the pixels)

Vector Graphics :
pixels only ~~cos~~ cause prob. during rasterization (when a vector image is converted to pixels for display)
resolution-independent image ⇒ approx.

4. Coord. Sys. Conversion.

new X = newLeft +
((old x − old Left) / (oldRight − oldLeft)) * (newRight − newL)

new Y = newTop +
((old Y − old Top) / (old Bottom − old Top)) * (newBottom − newTop)

5. Aspect Ratio :

ratio of width to height.

abs [(right − left) / (top − bottom)]

* mismatch → distortion.

✓ preserve aspect ratio

2.1.4 Color Models

1. Color Components :

RGB Model : red, green, blue.

intensity : 0 to 1 (max).

2. CMYK :

cyan, magenta, yellow, black.

* common for printers.

3. 24-bit color & 32-bit color :

8-bit can represent $2^8 = 256$ dif' values. 0 → 255

R

G

B

A Alpha → transparency = 0 fully transparent (invisible)

= 1 opaque.

* alpha blending.

new_color = (alpha) * (foreground) + (1 − alpha) * (background)

$\alpha \cdot$ color + $(1-\alpha) \cdot$ background

2.2 Shapes

1. Lines.
   ① round cap v. square cap. v. no cap.
   when 2 lines joining &
   line endings.

2. Rectangles.
   ① specify w/ 2 pts.
   - endpts of one diagonals

eg. fillRect(3,2,5,3) upper left corner (3,2) width 5 height 3.

3. Stroke & Fill.
   ① Stroke:
   drag a pen along the line / boundary.
   ② Fill:
   color all pts contained inside the shape.
   * winding no. *.
   how many times the shape winds around the pt.
   in the pos. direction. (counterclockwise).
   Rule A: color non-zero region
   B: color odd region.

3. Polygons, Curves, Paths.
   ① Polygon:
   defined by a list of its vertices.
   1) Regular Polygon:
   all sides same length.
   2) Convex Polygon:
   whenever 2 pts are inside or on the polygon,
   the entire line segment b/t the 2 pts is also
   inside or on the polygon.

Draw:

createPath() — start new path
moveTo(x, y) — move pen w/o drawing
lineTo(x, y) — draw line from current pen to (x, y)
closePath() — draw line from current pen to starting pt & end

② Bezier Curve:

defined by parametric polynomial eq.

eg. cubic Bezier Curve:

2 endpts of the segment & 2 ctrl pts.

2.3 Transforms

1. Viewport:

the rectangle made of pixels, w/ its natural pixel coor.
where an image will be displayed.

2. World Coord.:

used to define a set of geometric objects in often a real-no. coord. (not int pixels)

⇒ make the scene/world.

3. 2D Transforms:

$$x1 = ax + by + e$$
$$y1 = cx + dy + f.$$

(x, y) old coord. ⟶ $(x1, y1)$ new coord.

* Affine Transform:

a transform of the form

$$T(x, y) = (ax + by + e, cx + dy + f).$$

⊕ when applied to 2 parallel lines, transformed lines are also parallel.

⊕ if follow chain of affine transforms,
result is also affine transform. ☺.

4. Translation.

$x_1 = x + e$
$y_1 = y + f$

$$T_{e,f} = \begin{bmatrix} 1 & 0 & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+e \\ y+f \\ 1 \end{bmatrix}$$

$3 \times 3 \cdot 3 \times 1 \qquad 3 \times 1$

5. Rotation. * r is in radians.

$x_1 = \cos(r) \cdot x - \sin(r) \cdot y$
$y_1 = \sin(r) \cdot x + \cos(r) \cdot y$.

$$R = \begin{bmatrix} \cos(r) & -\sin(r) & 0 \\ \sin(r) & \cos(r) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(r) \cdot x - \sin(r) y \\ \sin(r) x + \cos(r) y \\ 1 \end{bmatrix}$$

6. Scaling

$x_1 = ax$
$y_1 = by$

$$S = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ 1 \end{bmatrix}$$

when $a=b$, uniform scaling $\Rightarrow$ w/o distortion.

$$T_{a,b} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \quad S_{a,b} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_d = \begin{bmatrix} \cos(d) & -\sin(d) & 0 \\ \sin(d) & \cos(d) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

objects constructed from sub-objects :) .

2.4 Hiearchical Modeling .

a complex object can be made up of simpler objects, which can in turn be made up of even simpler objects & so on, until it bottoms out w/ simple geometric primitives that can be drawn directly.

2.5 Java Graphics 2D

1. Real numbers :
   ① Double 64-bit
   - ⊕ more accurate .
   - ⊕ easier to use in Java
   - ⊖ more memory required .

   ② float : 32-bit
   - ⊕ less space
   - ⊕ generally enough accuracy !! .

* A pt is NOT a shape *
can't fill / stroke it .

2. Transforms :
   g2.scale(sx,sy)
   g2.rotate(r)
   g2.rotate(r, x, y)   about pt (x,y)
   g2.translate(dx, dy) .

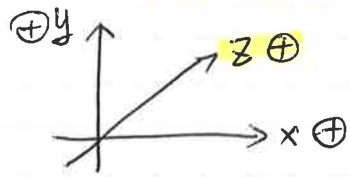
3. Off-screen canvas :
   work w/ images not visible on screen .
   eg. BufferedImage :
   represent a region in memory where you can draw, in exactly the same way that you can draw on the screen .

* <Polyline> :
similar to <polygon> but leaves out last line from final vertex to starting vertex .

Ch3 OpenGL 1.1 : Geometry 3D.

3.1 Shapes & Colors

1. 3D space :
   coord. sys. x, y, z from -1 to 1.
   Default : left-hand coord. sys.



2. OpenGL Primitives : built-in to the language.
   * points, lines, triangles } defined by vertices.
   glVertex3f(x, y, z).

3. Colors :
   glColor3f(r, g, b);
   glColor4f(r, g, b, a);
   ① Transparency :
   Default - turned off.
   glEnable(GL_BLEND);
   glBlendFunc (GL_SRC_ALPHA , GL_ONE_MINUS_SRC_ALPHA)

* colors are associated w/ individual vertices , NOT shapes * .
If dif' vertices w/ dif' colors ,
default interpolation of colors .
* colors must be specified b/f call to glVertex ~ .

glColor3d (0, 0, 0); ⇒ black ☺. default.

Clear the color buffer / drawing area :
glClearColor (r, g, b, a);

4. The Depth Test ✲

① Painter's Alg :

draw objects in order from back to front .

⊕ can do transparency

⊖ fail to address hidden surface prob .

② Depth test w/ depth buffer :

- depth :

distance from viewer to object . [0 - 1]

greater depth => further away .

✲ an object w/ smaller depth hides an object w/ larger depth ✲

- store a depth value for each pixel .

(in depth buffer)

⊕ solve hidden surface prob .

⊖ No transparency

⊖ confused on objects w/ same depth value

⊖ limited bits & limited accuracy .

Default - turned off .

glEnable (GL_DEPTH_TEST) .

0 = min distance

1 = max distance .

⊖ outside the range ⟶ NOT visible .

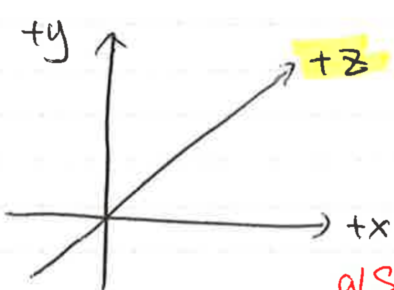
3.2 3D Coord. & Transforms

1. 3D Coord. Sys.

a way of assigning no. to pts .

projection & shading gives 2D -> 3D illusion .

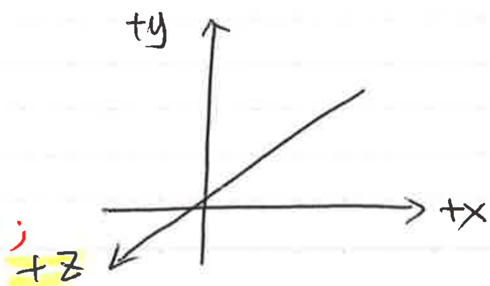
Default left-hand sys .



Scale (-1)
on z-axis
(1, 1, -1)
to get

glScalef(1, 1, -1);

conventional
right-handed sys .



2. 3D Transforms.

① Rotation:

about a line / axis of rotation.

- specified by 3 no. (ax, ay, az) not all $\emptyset$.

axis is through origin (0,0,0) & (ax, ay, az).

w/ an angle of rotation (degrees).

- Right-hand rule:

pt thumb in direction of axis from (0,0,0) to (ax, ay, az); the direction of rotation for pos. angles is the direction which fingers curl.

* only works in right-hand coord. sys.

② Scaling:

glScalef (sx, sy, sz)

(1, 1, -1) => reflect about xy-plane.

③ Translation:

glTranslated (dx, dy, dz).

* Transforms are applied to objects that are drawn after the transformation func. is called. & in opposite order of their appearance in code * ?.

3. Hierarchical Modeling.

use a stack of transforms.

① glPushMatrix():

save a copy of current matrix

② glPopMatrix():

restore the copy.

-> make sure:

current transform does not carry over to objects drawn later.

3.3 Projection & Viewing

1. Projection transformation :
   transformation from eye coord. to clip coord.

2. Viewport transformation :
   transform clip coord. to fit the viewport .

$$\begin{tikzcd} \text{object coord.} \arrow[r, "\text{modeling transform}"] \arrow[rr, bend right, "\text{modelview transform}"'] & \text{world coord.} \arrow[r, "\text{viewing transform}"] & \text{eye coord.} \arrow[r, "\text{projection transform}"] & \text{clip coord.} \arrow[d, "\text{viewport transform}"] \\ & & & \text{device coord.} \end{tikzcd}$$

3.4 Polygonal Meshes & glDraw Arrays

1. Polyhedron :
   represented exactly (w/ primitives pts, lines, polygons).
   ⊖ Curved surfaces are only approx.
   ↳ via a polygonal mesh :
   a set of polygons connected along their edges.

2. Indexed Face Sets (IFS) ⊕ useful when hard to compute vertices .
   ① data :
   a list of all vertices giving coord. of each vertex
   ② order :
   arbitrary for vertices .
   ③ Faces :
   front : counter-clockwise vertices order
   back : clockwise order .

front back .

3.5 Linear Algebra

1. Vector :

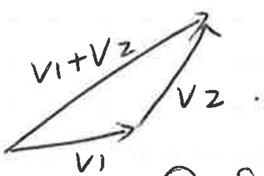
length & direction .

① unit vector :

vector of length 1 .
obtained from normalizing any vector
(divide vector by its length) .

② Addition :

$v1 = (x1, y1, z1)$
$v2 = (x2, y2, z2)$
$v1 + v2 = (x1+x2, y1+y2, z1+z2)$.



③ Scalar Multi :

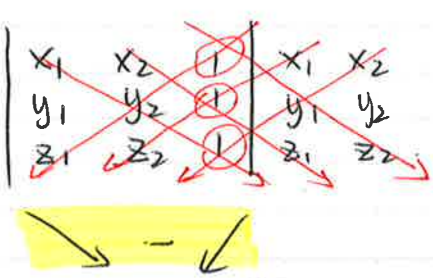
$v = (x, y, z)$
a is a no .
$a \cdot v = (ax, ay, az)$.

④ Dot Product :

$v_1 = (x1, y1, z1)$
$v_2 = (x2, y2, z2)$
$v_1 \cdot v_2 = x1 \cdot x2 + y1 \cdot y2 + z1 \cdot z2$
→ result is a number (not vector) .
$\cos(\text{angle}) = v_1 \cdot v_2 \,/\, (|v_1| * |v_2|)$
angle - b/t v_1 & v_2 .

※ 2 non-zero vector are ⊥ iff dot prod. is zero .

⑤ Cross Product .



$$v_1 \times v_2 = (y_1 \cdot z_2 - y_2 z_1,\; x_2 \cdot z_1 - x_1 z_2,\; x_1 \cdot y_2 - x_2 y_1) .$$

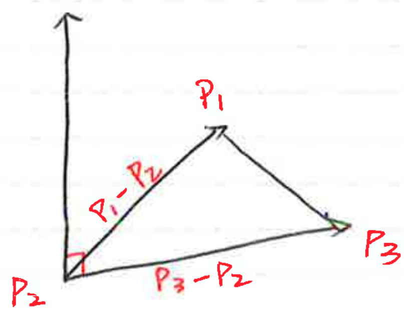
⑥ Normal Vectors.

If P_1, P_2, P_3 are vertices of a polygon,

~~$(P_3 \times$~~

$\perp$ $(P_3 - P_2) \times (P_1 - P_2)$ is the normal vector

$(P_3 - P_2) \times$
$(P_1 - P_2)$.



eg. $P = (1, 1, 1)$
$Q = (1, 2, 0)$
$R = (-1, 2, 1)$

$V_1 = Q - P = (0, 1, -1)$
$V_2 = R - P = (-2, 1, 0)$
$V_1 \times V_2 = (1, 2, 2)$. $\perp$

2. Matrices & Transformations:

① glTranslatef(tx, ty, tz).

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

② glScalef(sx, sy, sz)

$$\begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

③ glRotatef(d, 1, 0, 0)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(d) & -\sin(d) & 0 \\ 0 & \sin(d) & \cos(d) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

glRotatef(d, 0, 1, 0)

$$\begin{bmatrix} \cos(d) & 0 & \sin(d) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(d) & 0 & \cos(d) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

glRotatef(d, 0, 0, 1)

$$\begin{bmatrix} \cos(d) & -\sin(d) & 0 & 0 \\ \sin(d) & \cos(d) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ch4. Light & Material

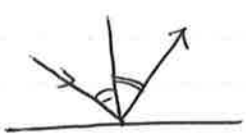
4.1 Lighting

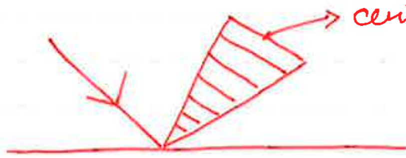
1. Default – disabled.
   glEnable(GL_LIGHTING)
   → turn it on.

   * material:
   is demonstrated as how it interacts w/ light.

2. Light Reflections

   ① Specular Reflection:
   - an incoming ray of light is reflected from the surface intact.
   - incidence angle = reflection angle.
   - specular highlights:
     even if the entire surface is lit, viewer can only see reflection of light source at those pts where angle is correct
     → cone of light in reality



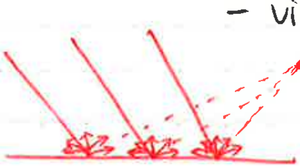


center w/ max reflection.

* duller surface gives wider cones.

   - shininess: [0 - 128]
     decides the size & sharpness of specular highlights.
     0 – largest specular highlight (eg. white circle)
     128 – smallest.

   ② Diffuse Reflection:
   - an incoming ray of light is scattered in all directions (equally if ideal).
   - viewer see the reflected light from all pts on surface.



3. Light Component:

some absorbed & some reflected diffusely & some specularly.

* color of material :
  - shows by the degree it reflects light of dif' wavelengths

  ① specular color => color of specular highlights.
  ② diffuse color => basic color.
  ③ emission color
  ④ ambient color.

4. Ambient Light:

a general level of illumination that does not come directly from a light source.

ambient color : how it reflects ambient light.

5. Emission Light :

emitted by the material itself.
→ allow the object to be seen w/o any light source.

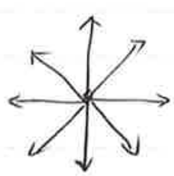
emission color : black if no emission.

* Color : really means reflectivity * . (for material)
intensity (for light).

6. Light Properties.

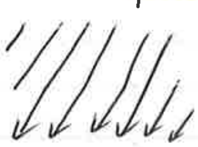
① Point light :

emits light in all directions from the pt.



② Directional light :

all light comes from same direction & all rays are parallel



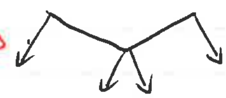
eg. the sun.

✻ color of material → reflectivity
color of light → intensity.

③ Light Source :
{ ambient color
diffuse color
specular color.

7. Normal Vectors :
a non-zero vector ⊥ to a surface at a given pt.
✻ OpenGL :
normal vectors are assigned only to vertices of primitives ✻.

① Flat Shading :
normal vectors ⊥ ~~see~~ ~~vertices~~ polygon surfaces
sharp edges.



② Smooth Shading :
normal vectors ⊥ to curved surface

✻ Specification : Right-hand rule ✻.
If you curl fingers in direction of order of vertices of the polygon, direction of thumb = direction of normal vector.

8. Equations :
R, G, B colors.
[0, 1]
values greater than 1, replaced by 1.

9. Material :
glMaterialfv(GL_FRONT_AND_BACK,
GL_DIFFUSE,
gold);

10. Define normal vectors :
✻ must be specified b/f glVertex*
glNormal3f(0, 0, 1); // default val.

11. 8 Light Sources :

(R, G, B, A) [0.0 → 1.0]

↳ not used for anything.

① position :

(x, y, z, w).

- Directional light :

  w = 0.

  ray shines from (x, y, z) to origin.

- Point light :

  w ≠ 0

  (x/w, y/w, z/w) is the loc. of light.

- Default at

  (0, 0, 1, 0).

② Global Ambient light :

default is black (0, 0, 0, 0). RGBA.

4.3 Textures

1. Texture :

some variation from pixel to pixel w/in same primitive.

① specified for each vertex

(similar to normal vector).

② must be b/f glVertex.

※ Texture is part of the attribute of the vertex.

Every vertex of a primitive needs a diff' set of texture coord.

Computer Graphics Midterm Review   Oct 1. 2019

Ch1

raster graphics grid of pixels w/ numerical color values for each
vector . as geometric shapes .
frame buffer block of mem to store colors for pixels .
display list . list of geometric shapes .
pixels has infinite pts .
geometric shapes .
geometric transforms:
- scaling - set the size by a factor
- rotation - set the orientation by rotating at an angle
- translation - Set the position by displacement.

Rasterization:
assigns colors from 3D graphics to 2D image .
approx. of the ideal image .

Ch2.
2D .

pixel coor . (col, row) int.
A pixel has many pts .

Aliasing :
multiple real no. coor. map to the same int pixel coor.

Antialiasing :
when a pixel is only partially covered by a shape,
the color of that pixel should be a mixture of the
color of the shape & background .

raster ← ⊖ machine dependent , wide range of pixel sizes .
✓ ~~[illegible]~~ vector graphics ✓ only prob.. when rasterization .
⇒ resolution - independent .

real no. coor.

setCoordinateSystem(L, R, B, T). ratio.

newX = newL + ((oldx - oldL)/(oldR - oldL)) * (newR - newL)

newY = newT + ((oldY - oldT)/(oldB - oldT)) * (newB - newT)

Aspect Ratio:

ratio of width to height.

RGB models. 8 bit/color. 24-bit color. (0-255).

CMYK models.

HSV/B - hue, saturation, lightness/brightness.

RGBA - alpha. transparency. 32-bit.

Stroke: drag a pen along its boundary /outline.

Fill: color all pts contained inside.

when intersect itself ← winding no.

Convex polygon:

whenever 2 pts are inside the polygon, the entire line segment b/t those pts is also inside the polygon.

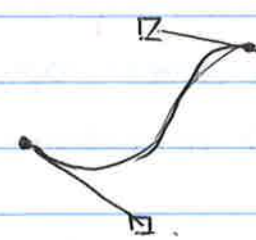
createPath()

moveTo(x, y)

lineTo(x, y)

closePath()

Bezier Curve: Control pts



cubicCurveTo(cx1, cy1, cx2, cy2, x, y).

Transforms.

viewport : the rectangle of pixels where an image is displayed.

world coor: x, y, z.

2D: world lies in a plane. $\Rightarrow$ view window. $\xrightarrow{\text{map}}$ viewport

$x1 = ax + by + e$

$y1 = cx + dy + f$

$(x, y) \Rightarrow (x1, y1)$.

Affine transform :

when applied to 2 parallel lines,

the transformed lines are also parallel

Translation.

$$\begin{cases} x1 = x + e \\ y1 = y + f. \end{cases}$$

Rotation. r-radians

$$\begin{cases} x1 = \cos(r)\,x - \sin(r)\,y \\ y1 = \sin(r)\,x + \cos(r)\,y. \end{cases}$$

Scaling

$$\begin{cases} x1 = ax \\ y1 = dy. \end{cases}$$

uniform scaling when $a = d$ w/o distortion.

$$T_{ab} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad S_{ab} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad R = \begin{bmatrix} \cos(d) & -\sin(d) & 0 \\ \sin(d) & \cos(d) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

translate. Scaling Rotation.

Java Graphics 2D . pixels

int coor. are defined to refer to lines b/t pixels .

fillRect (x , y , width , height) . upper left (x,y) .

JPanel : JFrame :

drawing surface (rectangular area on screen) .

paintComponent() method . •pack() shrink-wrap

↳ not called by user .

repaint() → call paintComponent()

BufferedImage . — off-screen canvas .

g2.scale (sx, sy) . .

g2 . rotate (r)

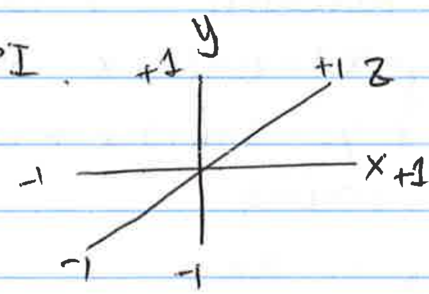
g2. rotate (r, x, y)

g2. translate (dx, dy)

Ch3 . awt & swing

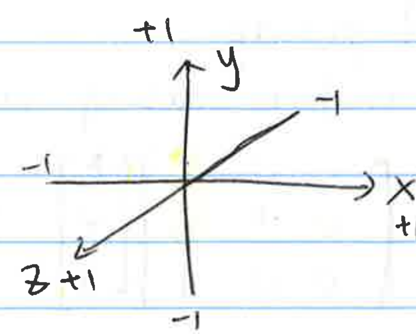
OpenGL : low-level graphics API .

1. default : left-hand coor. sys.



↓ scale z-axis by -1

to right-hand coor. sys .



2. Primitives : pts, lines, triangles.
built-in to the lang.

vertex 3D (x, y, z).

GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP.
↓
connect 1st to last v.

line width in pixels → NOT subject to scaling.

GL_Triangles, GL_triangle_strip, GL_triangle_fan.
new v ← prev2 vertices
new v ← prev v, 1st v.

Color: Transparency disabled by default.
3f(
3d(
unsigned: 3ub(
* OpenGL interpolate colors of vertices.
glBegin(GL_Triangles); color b/f vertex.
⋮
glEnd();

clear → color buffer. → limited bits & accuracy.

* Depth Test. * (depth buffer).
⊕ solve hidden surface prob.
depth: distance from viewer to object. 0–1
greater depth, farther away.
depth value for each pixel.
⊖ NO transparency
⊖ confused if same depth value.

Painter's alg. draw objects in order from back to front.

⊖ fail on hidden surface prob.

⊕ handle transparency,

3D. Transforms:

rotation about an axis defined by a pt & origin (0,0,0).

Direction of rotation:

right-hand rule,

pt thumb from (0,0,0) to (ax, ay, az).

fingers curl → + angles,

Do a 2D trans in 3D:

set z to ∅ for translation

1 for scaling & rotation,

$$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$\underset{(d,1,0,0)}{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(d) & -\sin(d) & 0 \\ 0 & \sin(d) & \cos(d) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \underset{(d,0,1,0)}{R} = \begin{bmatrix} \cos(d) & 0 & \sin(d) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(d) & 0 & \cos(d) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\underset{(d,0,0,1)}{R} = \begin{bmatrix} \cos(d) & -\sin(d) & 0 & 0 \\ \sin(d) & \cos(d) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$