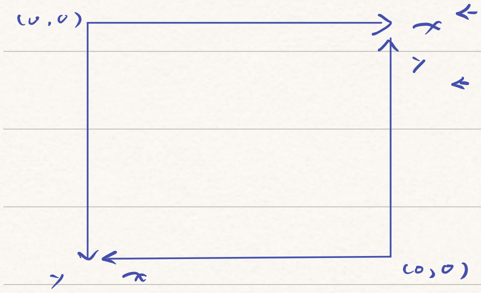
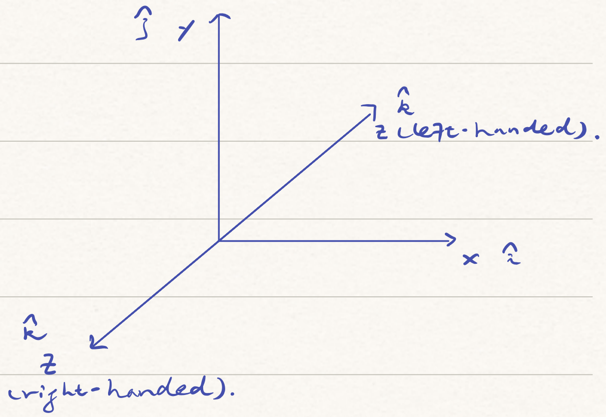
3.1 Displays & Windows



← left-handed sys

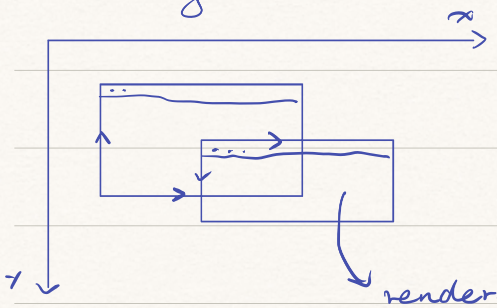
← Right-handed sys.



user space (defined by library/
device-based)

these don't have to be the same.

Windowing



each window would have ies own rendering system. So we only need to focus on one system.

rendering contex : 1:1 correspondence with their windows. Alive portion

glfw

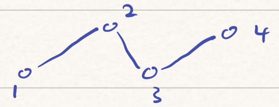
glut

Primitnes : things for rendering; associates with setting and parameters about how to render

Vertices: points in space $v = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Line segment

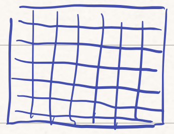
polyline: collection of line seg,



with vertices that head is the tail of another vertex

point: tiny circles with particular raidus. In fact, small squares.

Raster image: 2D grid of pixels, array of colors,



.bmp format

Polygon: closed shapes made of straight line segments

Simple Polygon: No intersepting edges

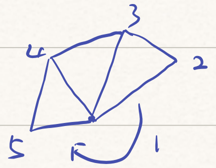
Convex : All interior angles are less than or equal to 180°

Concave : At least one interior angle > 180°

Polys => triangle

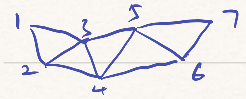
Any quad is two triangles => Any polygon is a collection of triangles

Triangle fan : triangles sharing one point

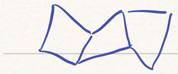


n vertices => n-2 triangles

Triangle strip n vertices -> n-2 triangles

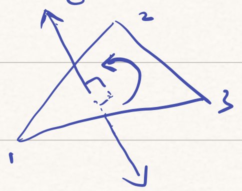


Quad Strip : similar with tri strip



n vertices -> $\frac{n-2}{2}$ quads

Winding Order: if we have a triangle, in a default counter-clock winding order is the front side



depend on vertices labelled order

Normal is coming out of front side.

we use n-1 object to describe n dimension thing

Face-culling: we only have to rendering things that are visible

OpenGL. each contact have a global state (state machine)

every api call as permanent state changes, until you explicitly remove it.

Client / Server : api is the seperator between client side and server side

side ↓ your program

server side ↓ state machine / render target / output device

Data need to be explicitly passed to server

OOD   Vulcan , DX12
↓
High-performance

Getting data to server:

immediate mode: render data is send every fram from client to server. which is characterized as GL begins, GL ends.

glBegin()

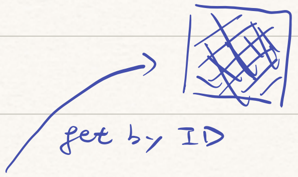
glVertex3f   ← immediate rendering

glEnd()

retained mode: data is send once, persists on the server side

glDrawArrays()

glDrawElements()



Steps to graphics

1. create a graphics context (runtime portion - server)
2. create a window / a target of the renderring
3. specify the data to draw
4. Draw in a loop

```
while (! window Closed ()) {
```