

Animation design patterns

Design Pattern : re-usable software design that solves a common problem

API: Application Programmer Interface

Game Loop Pattern

→ Implements an interactive application

```
while (!timeToQuit)  
    processInput()  
    updateApp()  
    display()
```

implements one frame
framerate depends on
speed of each loop
iteration

→ Typically, the application "customizes" what happens
in the game loop
how?

Patterns for responding to game loop events

Listener pattern: an obj is stored & its methods are called when "something" happens. (aka Observer / Subscriber)
ex. Swing components in JAVA

Callback fn: a ptr to a fn (or method) is stored & then invoked when "something"

ex. Javascript, GLFW

[
Facade pattern: an interface to hide the details of the game loop
& only provide high-level "hook"
ex. agl/window.cpp atkui/framework.cpp

framework.cpp

To build our own demos, override `atkui::Framework`

- override `setup()` to do custom initialization
- override `scene()` to update + draw objects each frame

note: any data that is part of the application should be a member of your framework subclass

Event-based architecture

Game loop pattern is an example

Events: initialization, draw, mouse clicks, key input, ...

Organize application in terms of events

for us: just need to design

what do we need to do for initialization?

" " each frame?

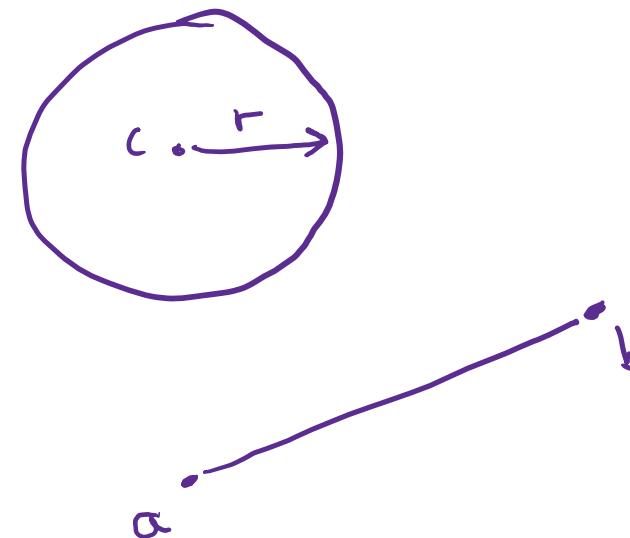
" " in response to mouse & key events?

Drawing

→ all drawing should be done in `Scene()`

Primitives :
 sphere
 → center c
 → radius r

line
 → 2 endpoints



Colors

RGB triplets, where each component varies from 0 to 1

The diagram illustrates an RGB triplet as a vector. The vertical axis is labeled "red" at the bottom. The horizontal axis is divided into two segments: "green" on the left and "blue" on the right. Three arrows point from the labels "red", "green", and "blue" to the corresponding components of the vector.

EX red $(1, 0, 0)$
green $(0, 1, 0)$
blue $(0, 0, 1)$

black $(0, 0, 0)$
white $(1, 1, 1)$

We sometimes use a 4th component, alpha, which controls transparency

0 : invisible
1 : opaque

→ note: blend must be enabled for transparency to work

RGBA

Points + Directions

Vector is a n -tuple of real numbers ($n = 2, 3, 4$)

ex. 3D point $p = (0, 2.3, 7)^T$ ← all vectors/ppts
RGB color $c = (\frac{1}{2}, 1, \frac{1}{10})^T$ are column
vectors!

Important to distinguish between points & directions!
defn. point: specific location in space
ex. address, a pin in a map

direction · has no location in space
vector
ex. north, east, left, up

Points and directions are interpreted differently

$$\text{ex. } A \text{ pt} + \text{direction} = \text{pt}$$

$$\text{ex. } A \text{ pt} - \text{pt} = \text{direction}$$

$$\text{ex. } A \text{ direction} + \text{direction} = \text{direction}$$

we use `glm::vec3`
to implement both

Coordinate System (aka Coordinate Frame)

Meaning of a pt or vector depends on the coordinate system.

Defined by

right-handed
system

→ origin $(0, 0, 0)$

→ axes : directions corresponding to each component

$\hat{i} (1, 0, 0)$ x - coordinate

$\hat{j} (0, 1, 0)$ y - coordinate

$\hat{k} (0, 0, 1)$ z - coordinate

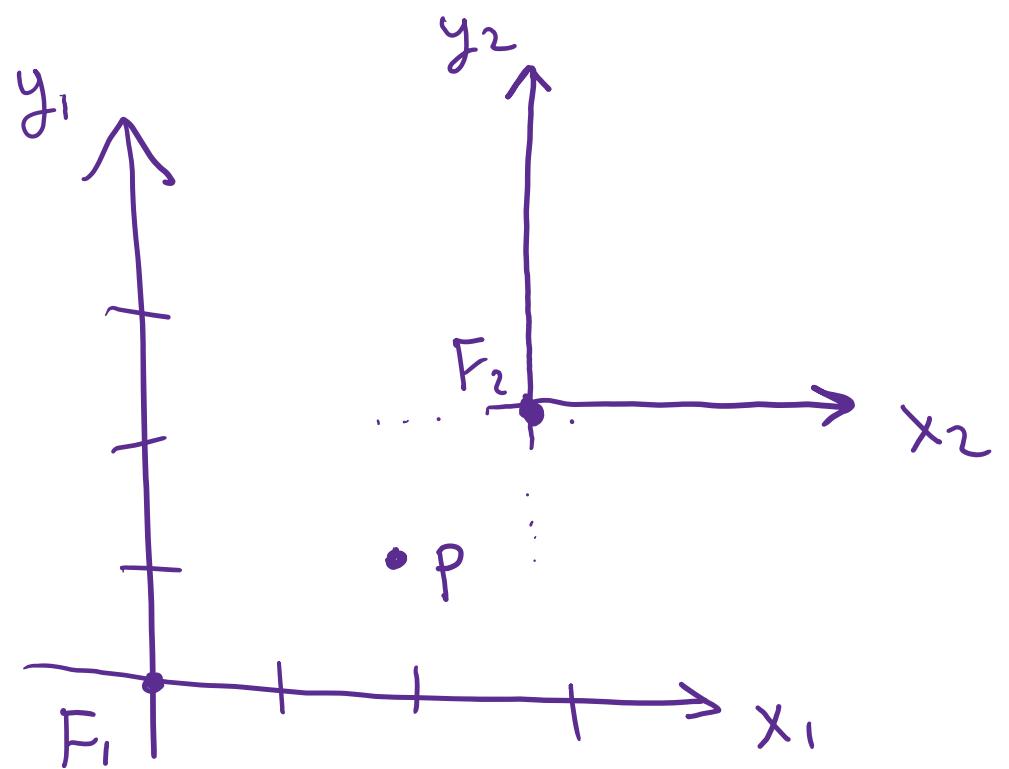
Example

F_2 is located at $(3, 2)^T$ w.r.t to F_1

What is the coordinate of P

w.r.t F_1 ? $(2, 1)^T$

w.r.t F_2 ? $(-1, -1)$



Positions, velocity, time

Velocity is an n-tuple where each component is a rate of change

Ex. $\text{vel} = (-1, 3, 4)^T \rightarrow$ move -1 units/s along x
3 units/s along y
4 units/s along z

Ex. Suppose we start at position $(0, 1, -4)^T$ + and we move by vel. Where is the point after 2 s?

$$\begin{aligned} p &= \begin{pmatrix} 0 \\ 1 \\ -4 \end{pmatrix} + \begin{pmatrix} -1 \\ 3 \\ 4 \end{pmatrix} \times 2 = \begin{pmatrix} 0 - 2 \\ 1 + 6 \\ -4 + 8 \end{pmatrix} = \begin{pmatrix} -2 \\ 7 \\ 4 \end{pmatrix} \\ &= p_0 + vt \end{aligned}$$

Note: Speed is the magnitude of velocity
 $\|v\| = \sqrt{(-1)^2 + 3^2 + 4^2}$

Let's use velocity to implement a smoothly moving particle

Alg:

```
setup()  
    currentPos = vec3(0, 1, -4)  
    vel = vec3(-1, 3, 4)  
  
scene()  
    currentPos = currentPos + vel * dt()  
    drawSphere(currentPos) 100.0);  
  
data { vec3 currentPos;  
        vec3 vel; }
```

time since
the last
call
scene

part of
the framework

Note: framework also gives us
elapsedTime()