CMPE163 August 20 (Fri)
Organizational Meeting

1) Harry Li Email:
hua.li@sjsu.edu
(650) 400-1116 Text

Office: M, W. 3:40-4:40 pm.
Zoom ID + Pass code
is the Same as
what you have today.

Lecture Zoom Link sent to
the class today.

Note: Homework, Projects
Announcements will be made
in Class, posted on Line as
github
CANVAS, Submission of homework
projects will be
on CANVAS.

TextBooks + References (optional)

a. Unity Tutorial, 3D Graphics
GameDev. Engine

b. Other Optional TextBooks —
Reference Only.

Programming Languages + Software
IDE

1. Unity, Student OR Personal
Edition. → Karting GAME

2. Python for Graphics
Video, Version 3.6 or
higher.
Anaconda, Tool for
Python Programming →

3. C/C++ for 2D & 3D
Graphics, Videos.

4. C# for Interface to
Unity IDE.

5. OpenCV. Homework:
Installation of OpenCV.
In 2 Weeks Sept. 2nd (Th)

6. OpenGL Installation
of OpenGL. Homework:
Installation, and have it
ready By Next week
Aug. 26 (Th) Before
4:00 pm..

7. O.S. Ubuntu 18.04

Installation of Unity
By Aug. 26 (Th).
Before 4:00 p.m.

Grading Policy:
30% Projects, Homework etc.
30% midterm (ONE)
40% Final (Comprehensive)

Conduct of the Class

1° Lecture 2° Show + Tell

3° Form A team, 2-3 Person Team.

All homework, Coding have to be individual, however teamwork is encouraged, and be required

Projects, Homework: Assigned Projects. (3 projects)

plus A-Semester-Long project (Team) Project

a 2-3 person team;

b Proposal of A-Semester_Long Project;

c Progress Report & Presentation During class show + tell

d Final Presentation (PPT. Demo)

3 projects.

Project to Build 3D Animated Graphics.
+
Virtual Camera + Video

"GAME"-Like Environment

Robotics
Self-Driving.

August. 26 (Th)

Topics 1° Software Development Tool

2° Vector Graphics
2D Vector Graphics.

Reference Link: github/hualili

Software Tool: First, Unity up By Friday

OpenGL Installation on your Machine

Example: Running Unity. "Karting" GAME

Start the Unity

Step 1. On the Right hand UI. Interactive Tutorial Panel (Window)

Select / Go through 2 Tutorial

First. Tutorial — play the Karting GAME

Step 2. UI Editor

a Scene View Window

b Hierarchy Window

3D Graphics + Video

Hierarchy: "Everthing" Defined in this Window,

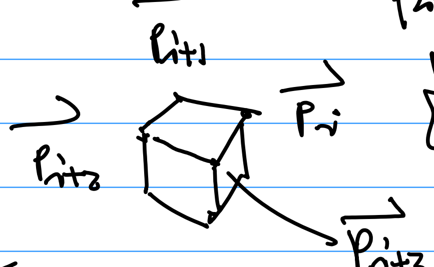
$\underline{a}$ PAN;
$\underline{b}$ Zoom In/Out, $\underline{c}$ Orbit Movement (Virtual Camera)

Use this platform to modify the "Karting" GAME. Removal of Some/all
- 3D Objects
- Re-Building 3D Scene. (3D World coordinate)

Introduction to 2D Vector Graphics.

2D: Dimensional Description

Vector: Vertices (Vertex) To Define Graphics Pattern(s)



$\{\vec{P}_i \mid i=1,2,\dots,4\}$

Fig.1. 3D

Vector → Vertex → Point

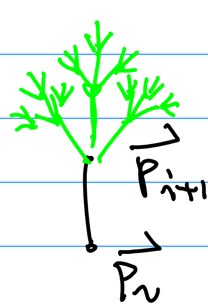


Fig.2 2D Vector Graphics

2D Vector Definition of a Line Segment

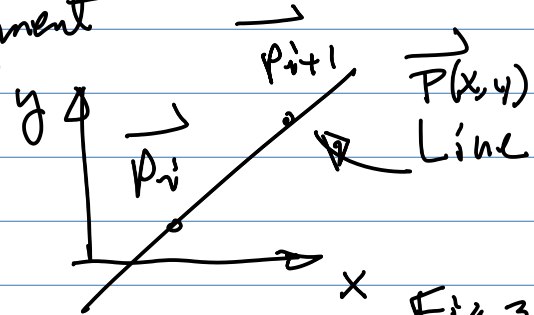


Fig.3

x-y Coordinate System

"Virtual" Display Coordinate System

Primitive Graphics

2 pts to uniquely define a line.
$\vec{P}_i$, $\vec{P}_{i+1}$

Notation

$\vec{P}_i$ Short Hand Notation

$\vec{P}_i(x_i, y_i)$, x_i-, y_i- Comp.

$\vec{P}_i(x_i, y_i) = (x_i, y_i)$ for Coding in C/C++, python, ...

To Define A line

(1) Direction of the Line

$$\vec{d} \triangleq \vec{P}_{i+1} - \vec{P}_i \quad \dots (1)$$

Ending pt. Starting pt

Eqn(1), Can be written as follows

$$\vec{d}(x_d, y_d) = \vec{P}_{i+1}(x_{i+1}, y_{i+1}) - \vec{P}_i(x_i, y_i) \quad \ldots (1\text{-}a)$$

For Coding purpose,

$$\begin{cases} x_d = x_{i+1} - x_i & (1\text{-}b) \\ y_d = y_{i+1} - y_i & (1\text{-}c) \end{cases}$$

Write C-code for the directional vector in Eqn(1-b), (1-c)

```
x_d[i] = x[i+1] - x[i] ; // for x-Comp of the directional vector
y_d[i] = y[i+1] - y[i] ; // for y-Comp. of the directional Vector.
```

... (1-d), (1-e)

② Need A pt to make an Unique Line

$$\vec{P}(x, y) = \vec{P}_i(x_i, y_i) + \lambda \vec{d}(x, y) \quad \ldots (2)$$

where λ is scalar

Physical meaning: $\vec{P}(x,y)$ Any pt. on the Line

$\vec{P}_i(x_i, y_i)$ A given pt (Known) on this Line

$d(x,y)$, A directional vector of the Line

Let $\lambda = 0$, $\vec{P}(x,y) = \vec{P}_i(x_i, y_i)$ starting pt.

From Eqn(2), $\vec{P}(x,y) = \vec{P}_i(x_i, y_i) + \lambda\left(\vec{P}_{i+1}(x_{i+1}, y_{i+1}) - \vec{P}_i(x_i, y_i)\right) \ldots (2a)$

Question: How to find the Ending pt from Eqn (2a) ?

if $\lambda = 1$

$$\vec{P}(x,y) = \vec{P}_i(x_i, y_i) + 1 \cdot \left(\vec{P}_{i+1}(x_{i+1}, y_{i+1}) - \vec{P}_i(x_i, y_i)\right)$$

$$= \cancel{\vec{P}_i(x_i, y_i)} + \vec{P}_{i+1}(x_{i+1}, y_{i+1}) - \cancel{\vec{P}_i(x_i, y_i)}$$

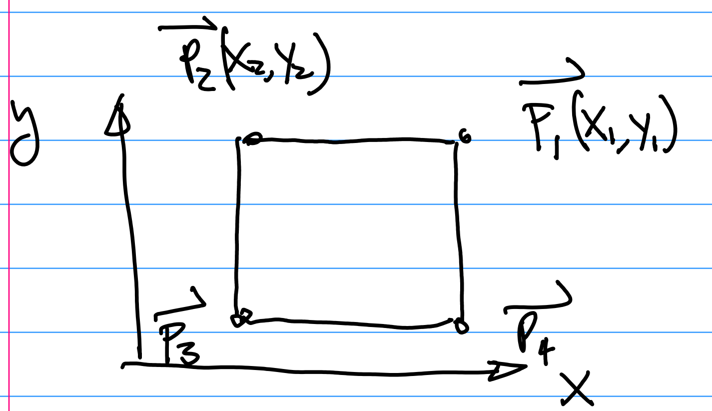
$$= \vec{P}_{i+1}(x_{i+1}, y_{i+1}) \text{ Ending pt.}$$

Screen Saver a collection of 2D Rotating Patterns. (Squares)

Example: Using Eqn (2a) to Create 2D Rotating Squares as a Screen Saver.

Define 2 vectors (pts)

Step 1. $\vec{P}_i(x_i, y_i)$, and $\vec{P}_{i+1}(x_{i+1}, y_{i+1})$



$\vec{P}_1(x_1, y_1) = (60, 60)$, $\vec{P}_2(x_2, y_2) = (10, 60)$

And to Define A Line in Parallel with $\vec{P}_1$ & $\vec{P}_2$

$\vec{P}_3(x_3, y_3) = (10, 10)$, $\vec{P}_4(x_4, y_4) = (60, 10)$

Connect $\vec{P}_2$ to $\vec{P}_3$, Similarly $\vec{P}_1$ to $\vec{P}_4$

Therefore, we have formed A square

Line Equation for Line (Top Line)

$$\vec{P}(x,y) = \vec{P}_1(x_1, y_1) + \lambda\left(\vec{P}_2(x_2, y_2) - \vec{P}_1(x_1, y_1)\right) \quad \ldots(3a)$$

Line for $\vec{P}_2(x_2, y_2)$ and $\vec{P}_3(x_3, y_3)$

$$\vec{P}(x,y) = \vec{P}_2(x_2, y_2) + \lambda_2\left(\vec{P}_3(x_3, y_3) - \vec{P}_2(x_2, y_2)\right) \quad \ldots(3b)$$

And for the other 2 Lines

$$\vec{P}(x,y) = \vec{P}_3(x_3, y_3) + \lambda_3\left(\vec{P}_4(x_4, y_4) - \vec{P}_3(x_3, y_3)\right) \quad \ldots(3c)$$

And

$$\vec{P}(x,y) = \vec{P}_4(x_4, y_4) + \lambda_4\left(\vec{P}_1(x_1, y_1) - \vec{P}_4(x_4, y_4)\right) \quad \ldots(3d)$$

These 4 equations define the Boundary of the Square.

From Coding Aspect:

From Simple Example (1-d), & (1-e)

Eqn (3a) becomes

$$\begin{cases} x = x_1 + \lambda(x_2 - x_1) & \ldots(4a) \\ y = y_1 + \lambda(y_2 - y_1) & \ldots(4b) \end{cases}$$

Define A buffer for x, And a buffer for y.

Each x_1, y_1, x_2, y_2 are also

Therefore, C/C++ Coding Implementation for (4-a), (4-b) can be done accordingly.

Homework: Install openGL on your machine, By Next Lecture, So we will use it for Rotating Squares implementation.