1/

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
Game Dev. 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 (P.P.T. 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: "Everything" Defined in this Window,

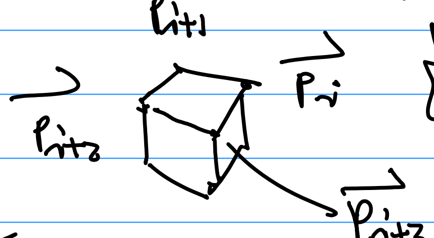
a. PAN;
b. Zoom In/Out, c. Orbit Movement (Virtual Camera)

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.

Dimensional Description → 2D

V → Vertices (Vertex) To Define Graphics Pattern(s)



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

Fig. 1. 3D

Vector → Vertex → Point
← ←

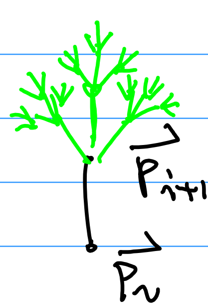


Fig. 2 2D Vector Graphics

2D Vector Definition of a Line Segment

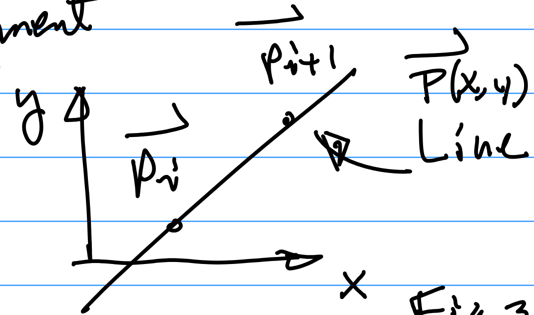


Fig. 3

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

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

Ending pt. Starting pt

Eqn(1), Can be written as follows

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

For Coding purpose,

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

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.
```

$\cdots (1-d), (1-e)$

(2) Need A pt to make an Unique Line

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

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

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

if $\lambda = 1$

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

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

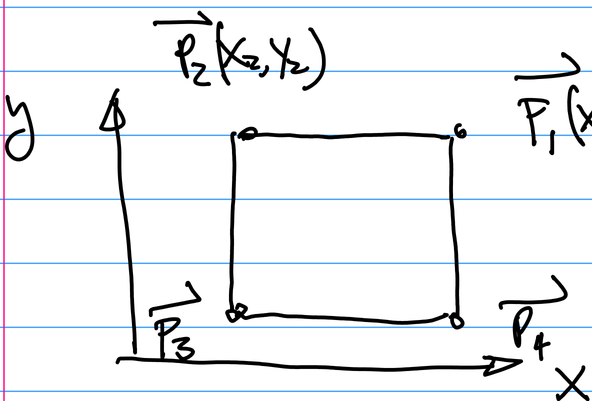
$$= \vec{P}_{i+1}(x_{i+1}, y_{i+1}) \text{ Ending pt.}$$

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.

Sept 2nd (Th)

Topics: 1° 2D Screen Saver Implementation;

Ref: github/hualili/opencv/

Homework: (To Be Submitted in 1 Week) Submission 4:00pm. Sept. 9th (Th) (1pt)

Visit homework Assignment on OpenGL, Source code .cpp has been posted.

Example: OpenGL CPP code

1° Create A program header template, Start your Program with this unified template

a. Program Name

b. Coded by

c. Date, d. Version

e. Status (Debugging, Release). f. Compilation and Built; g. Ref. (URL)

b. glBegin( );

⋮

glEnd( );

glClear( );

GL_POLYGON Keyword.

Vertex (pt)

Homework. GL_LINES

Modify the Sample code to draw a line with

$\vec{P_i}(x_i, y_i) = \vec{P_1}(x_1, y_1) = (50, 50)$

$\vec{P_{i+1}}(x_{i+1}, y_{i+1}) = \vec{P_2}(x_2, y_2) = (60, 100)$

Compile your program, run it. E-mail your Screen Capture, or 5 seconds Video Clips. → Submission in e-mail. Before Sept 9 4:00pm. (No point)

Create Rotating Squares for a Screen Saver.

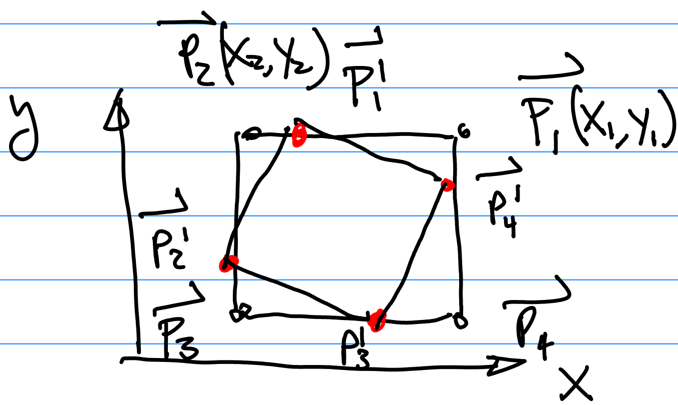


Fig 1. Note: $\vec{P_1}, \vec{P_2}, \cdots, \vec{P_4}$ are defined in A Counter Clockwise direction (Later in 3D Graphics we will do Hidden Line/Surface Removal)

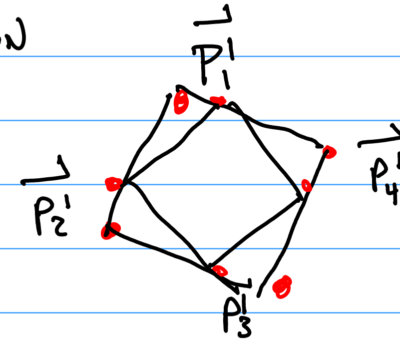


Fig. 2

Repeat the Same Process, however with New set of Points, $\vec{P_1'}$, $\vec{P_2'}$, $\vec{P_3'}$, $\vec{P_4'}$

Continue this process, we have:

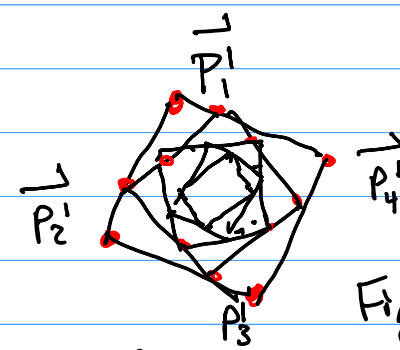


Fig. 3

To generalize this process, we introduce a superscript j as follows,

From Eqn (2a)

From Eqn (2-a),

$$\vec{P}(x,y) = \vec{P_1}(x_1,y_1) + \lambda\left(\vec{P_2}(x_2,y_2) - \vec{P_1}(x_1,y_1)\right)$$

Let $\lambda = 0.8$ ~~0.5~~

$$\vec{P}(x,y) = \vec{P_1}(x_1,y_1) + \lambda\left(\vec{P_2}(x_2,y_2) - \vec{P_1}(x_1,y_1)\right) \cdots (3a)$$

If $\lambda = 0.8$, $\vec{P}$ is 80% pt on the line formed by $\vec{P_1}$ & $\vec{P_2}$;

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

becomes

$$\vec{P}_i^{j+1}(x_i^{j+1}, y_i^{j+1}) = \vec{P}_i^{j}(x_i^{j}, y_i^{j}) + \lambda\left(\vec{P}_{i+1}^{j}(x_{i+1}^{j}, y_{i+1}^{j}) - \vec{P}_i^{j}(x_i^{j}, y_i^{j})\right) \quad \dots (1)$$

The Above Equation can be written in Explicit form (x-Comp, y-Comp)

For x-Comp.

$$\begin{cases} x_i^{j+1} = x_i^j + \lambda(x_{i+1}^j - x_i^j) & \dots (2a) \\ y_i^{j+1} = y_i^j + \lambda(y_{i+1}^j - y_i^j) & \dots (2b) \end{cases}$$

C/C++ Code

```
x_buf[i][j+1] = x[i][j] - lamda*(x[i+1][j]-x[i][j]);
y_buf[i][j+1] = y[i][j] - lamda*(y[i+1][j]-y[i][j]);
```

Sample code Example: github/hualili/openCV ... /1_line.cpp.

https://github.com/hualili/opencv/blob/master/ComputerGraphics_AR/F2018/1_line.c

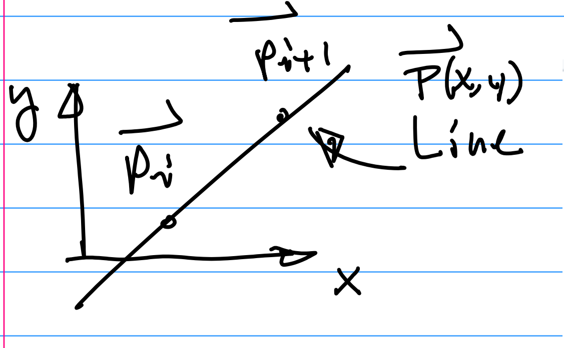


Fig.4

① Header

```
/*****************************************************
 * Program: line.c    Coded by: Harry Li            *
 * Version: x1.0;     status: tested;               *
 * Compile and build:                               *
 * gcc main.cpp -o main.o -lGL -lGLU -lglut         *
 * Date: Jun 5, 2014                                *
 * Purpose: Graphics Demo.                          *
 *****************************************************/
#include<GL/glut.h>
#include<stdio.h>
```

② Libraries

③ $\vec{P_1}(x_1,y_1)$, $\vec{P_2}(x_2,y_2)$, $\vec{P_1}(x_1,y_1) = (1,1)$, $\vec{P_2} = (-1,-1)$

```
void mydisplay()
{
float p1x=1.0f,p1y=1.0f;    //the window coordinates (-1.0, 1.0)
float p2x=-1.0f,p2y=-1.0f;
```

(4) Note: House Keeping for 2D Graphics

```
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
```

(5) { glBegin( );
glEnd( );

GL_LINES

```
glBegin(GL_LINES);
glVertex2f(p1x,p1y);
glVertex2f(p2x,p2y);
glEnd();
```

glVertex2f(x, y) → $P_i(x_i, y_i) = (x_i, y_i)$

Note: In your homework, please
2D Sample code,