CMPE163 August 20 (Fri)
Organizational Meeting

1) Harry Li E-mail:
hua.li@sjsu.edu
(650) 400-1116 Text
Office: M.W. 3:40-4:40 pm.
Zoom ID + Passcode
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 homwork
projects will be
on CANVAS.

Text Books + References (optional)
a. Unity Tutorial, 3D Graphics
Game Dev. Engine
b. Other Optional Text Books —
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. Open CV. Homework:
Installation of Open CV.
In 2 Weeks Sept. 2nd (Th)
→ 6. OpenGL Installation
of OpenGL. Homework:
Installation, and have it
ready By Next week
Aug. 26 (Th) Before
five pm..
→ 7. O.S. Ubuntu 18.04

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

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
3D Vector Graphics.
Reference Link: github/flyalibi
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 Movment (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 ⇑ Vertices (Vertex) To Define Graphics Pattern(s) ⇑ $\{\vec{P_i} \mid i=1,2,\dots,4\}$

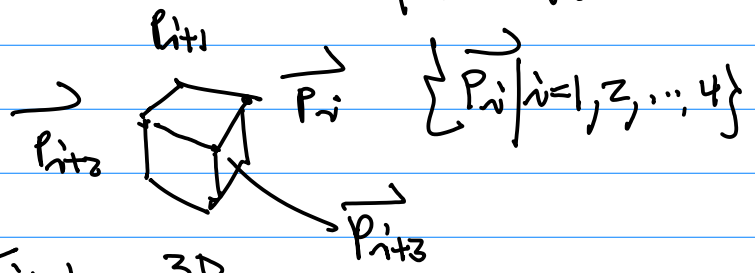


Fig. 1. 3D

Vector ⇒ Vertex ⇒ Point
⇐ ⇐

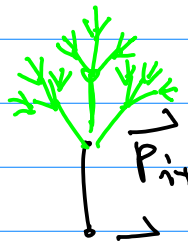


Fig. 2 2D Vector Graphics

2D Vector Definition of a Line Segment

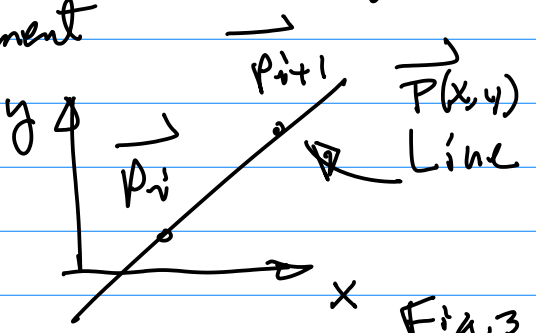


Fig. 3

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

$$\vec{d} = \vec{P_{i+1}} - \vec{P_i} \quad \dots (1)$$

Ending pt. Starting pt

Eqn(1), C 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)

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

if $\lambda = 1$

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

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

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

```
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.

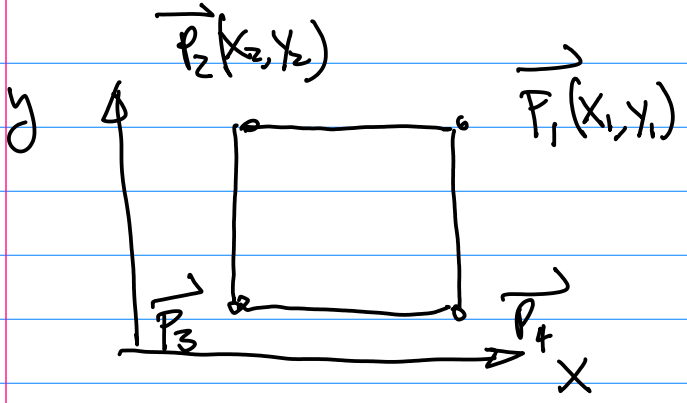
From Eqn(2), $\vec{P}(x,y) = \vec{P}_i(x_i, y_i) + \lambda \left(\vec{P}_{i+1}(x_{i+1}, y_{i+1}) - \vec{P}_i(x_i, y_i)\right) \cdots (2a)$

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 Sample/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(k=a), (k=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/fnwalidi/openGL/

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

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

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

Compile your program, run it.
E-mail your Screen Capture, or 5 seconds Video Clips.

Submission in e-mail. Before Sept 9 4:30pm. (No point)

Create Rotating Squares for a Screen Saver.

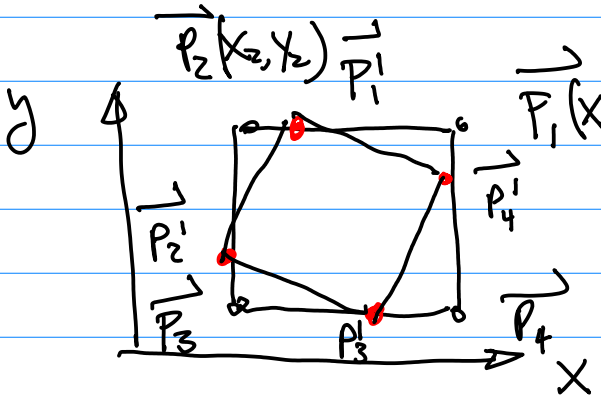


Fig 1. Note: $\vec{P}_1, \vec{P}_2, \cdots, \vec{P}_4$ are defined in A Counter Clockwise direction (Later in 3D Graphics we will do Hidden Line/Surface Removal)

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 $\lambda = 0.8$ $\quad$ ~~0.5~~

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

If $\lambda = 0.8$, $\vec{P}$ is 80% pt on the line formed by $\vec{P}_1$ & $\vec{P}_2$;

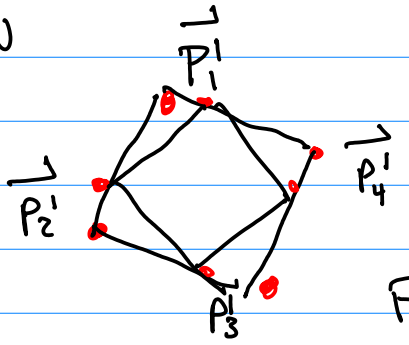


Fig. 2

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:

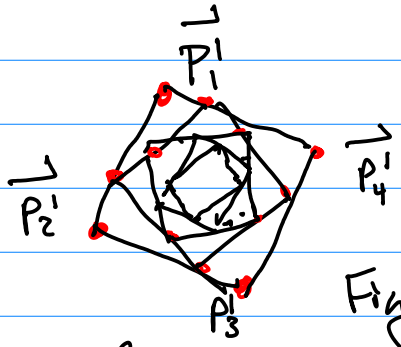


Fig. 3

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

From Eqn (2a)

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

becomes

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

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

For x-Comp.

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

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

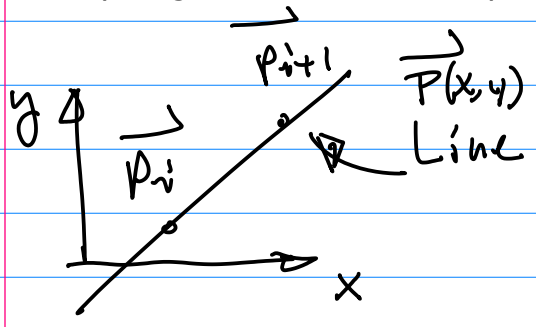


Fig.4

① Header

```
1  /*****************************************************
2   * Program: line.c   Coded by: Harry Li              *
3   * Version: x1.0;    status: tested;                 *
4   * Compile and build:                                *
5   * gcc main.cpp -o main.o -lGL -lGLU -lglut          *
6   * Date: Jun 5, 2014                                 *
7   * Purpose: Graphics Demo.                           *
8   *****************************************************/
9  #include<GL/glut.h>
10 #include<stdio.h>
```

② Libraries

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

```
11 void mydisplay()
12 {
13 float p1x=1.0f,p1y=1.0f;    //the window coordinates (-1.0, 1.0)
14 float p2x=-1.0f,p2y=-1.0f;
```

```
15  glClear(GL_COLOR_BUFFER_BIT);
16  glLoadIdentity();
```

(4) Note: House Keeping for 2D Graphics

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

```
17  glBegin(GL_LINES);
18  glVertex2f(p1x,p1y);
19  glVertex2f(p2x,p2y);
20  glEnd();
```

GL_LINES

glVertex2f(x,y) → $\vec{P}_i(x_i,y_i) = (x_i,y_i)$

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

Sept. 9 (Th)

1. Show-And-Tell
2. Today's Topics: 2D Vector Graphics for Screen Saver Application, Squares (Rotating Pattern), Trees

Show+Tell Ken, Bill. Patrick, Anh

Homework: Implementation of Rotating Squares Based on Eqn(2.4), on pp8.

Due A week from Today.

Show+Tell

Optional Homework: Write a script CS (C#) to Control your Design.

~ 2 weeks

Example: Creating a Tree By 2D Vector Graphics.

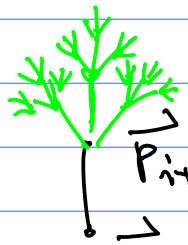


Fig. 1 2D Vector Graphics

Note: 1° The Levels of Iteration should be at Least 7 or Higher;

2° Random Function Generator to Allow Each tree to be placed at Random Locations. rand( );

3° Python Version Openbal for the Implementation is encouraged.

Description of the Algorithm:

1. You Design the Length of A Tree Trunck. By 2D Vectors. $\vec{P}_i$, $\vec{P}_{i+1}$

$\vec{P}_i$: Starting pt; $\vec{P}_{i+1}$ Ending pt.

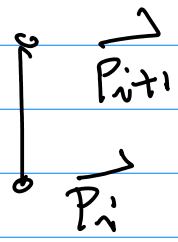


Fig 2

$\vec{P}_i(0,-10)$

$\vec{P}_{i+1}(0,0)$ ... (*)

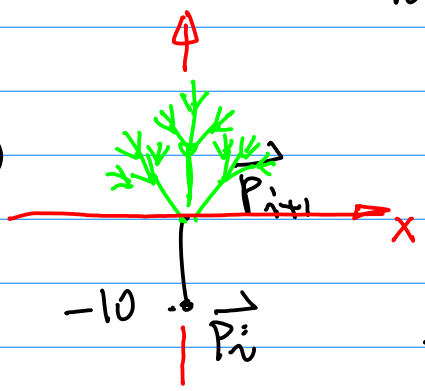
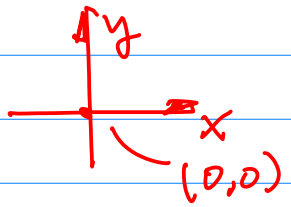


Fig 4

2. Generate Another (Next Level) Tree Branch → Main Branch (Same Direction) as its previous Level;

{ a. Same Direction

b. Reduction By 20%



From Eqn (1a), (1b), with the Given Condition (*), we have

$$\vec{P}'_{i+1} = \vec{P}_{i+1} + \lambda(\vec{P}_{i+1} - \vec{P}_i)$$

$$= (0,0) + \lambda((0,0) - (0,-10))$$

$$= (0,0) + \lambda(0,10) \quad \ldots (2)$$

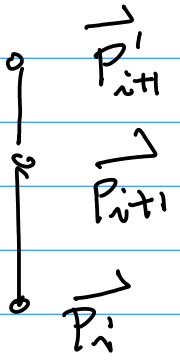


Fig 3

To find New Ending pt $\vec{P}'_{i+1}$, Let $\lambda = 0.8$, Substitute λ into the above equation

$$\vec{P}'_{i+1}(x'_{i+1}, y'_{i+1}) = \vec{P}_{i+1} + \lambda(\vec{P}_{i+1} - \vec{P}_i) \quad \ldots (1)$$

Starting Pt. ↑ Direction Vector of the previous Level

$$\vec{P}'_{i+1} = (0,0) + 0.8(0,10)$$

$$= (0 + 0.8 \times 0,\ 0 + 0.8 \times 10)$$

$$= (0,8)$$

$$\begin{cases} x'_{i+1} = x_{i+1} + \lambda(x_{i+1} - x_i) & \ldots (1a) \\ y'_{i+1} = y_{i+1} + \lambda(y_{i+1} - y_i) & \ldots (1b) \end{cases}$$

3. (Create the other 2 Side Branches (Left Branch, Right Branch)

Define A pt $\vec{P}_i(x_i, y_i)$
Rotate this pt. $\vec{P}_i(x_i, y_i)$ By α

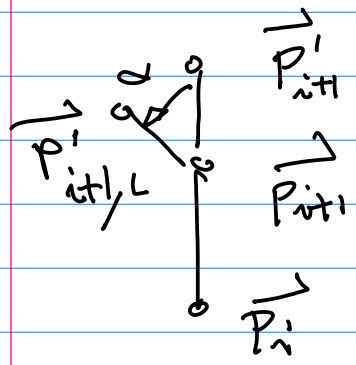


Fig 5.

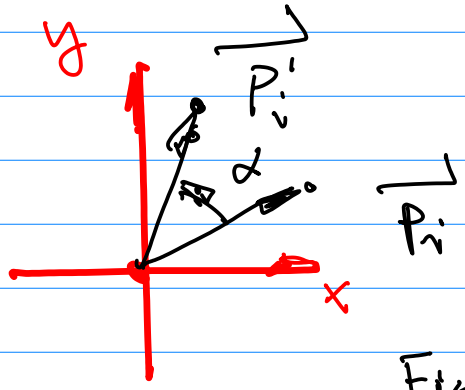


Fig 6.

Left Branch: Rotating the main Branch (at the Same Level) Counter Clockwise By α (Angle) Denote the New Branch as $\vec{P}'_{i+1,L}$

Note: Need a Newer Math. Formulation for this Rotation.

$$R_{3\times3} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \ldots(3)$$

$\vec{P}_i(x_i, y_i) = (x_i, y_i)$,

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

$\Downarrow$ Col. Vector

$$\vec{P}'_i(x_i, y_i) = (x_i, y_i)^T = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

$$\vec{P}'_{i+1} = \begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix}$$

For Simplicity,

$$\vec{P}_i(x_i, y_i) = \begin{pmatrix} X_i \\ Y_i \end{pmatrix},$$

$$\vec{P}_{i+1}(x_{i+1}, y_{i+1}) = \begin{pmatrix} X_{i+1} \\ Y_{i+1} \end{pmatrix}$$

CMPE163

A Pt $\overrightarrow{P_i} = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, Before the Rotation,

Pt. $\overrightarrow{P_i'} = \begin{pmatrix} x_i' \\ y_i' \end{pmatrix}$, After the Rotation

After the Rotation $\overset{?}{=}$ Before the Rotation,

Hence, we need Rotation matrix

Eqn (3),

Now, for the Branch to the Right,

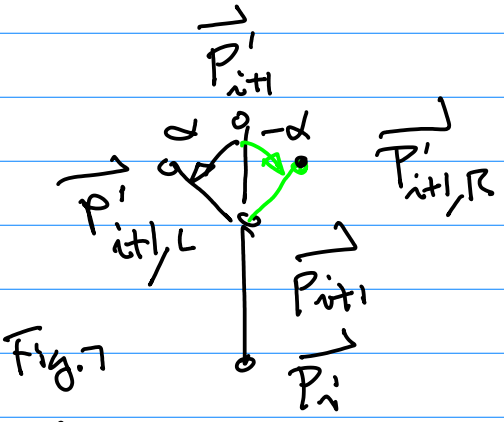


Fig. 7

After

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \quad \text{Before} \quad \cdots (3b)$$

Rotation

Therefore, $-\sin(-\alpha) = \sin\alpha$

$$\begin{pmatrix} x'_{i+1,R} \\ y'_{i+1,R} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x'_{i+1} \\ y'_{i+1} \\ 1 \end{pmatrix} \quad \cdots (5)$$

Suppose $\alpha = 30°$

$$\begin{cases} x_i' = x_i \cos\alpha - y_i \sin\alpha & \cdots (4a) \\ y_i' = x_i \sin\alpha + y_i \cos\alpha & \cdots (4b) \end{cases}$$

$$\begin{cases} x'_{i+1,R} = \cos\alpha \cdot x'_{i+1} + \sin\alpha \cdot y'_{i+1} & \cdots (6a) \\ y'_{i+1,R} = -\sin\alpha \cdot x'_{i+1} + \cos\alpha \cdot y'_{i+1} & \cdots (6b) \end{cases}$$

Note: 1° Positive α: α in Counter Clockwise direction;
Negative α: Clockwise

2° The Rotation in (3b) defines the Rotation w.r.t the origin of the x-y Coordinate System.

Team 1
- Christiana 1702,
- Anh 3807,
- Charles 0436
- Ken 3381

Team 2
- Christoffer 7230
- Patric 8001
- Anmol 9654

Team3 { Yong 4236
Zhonglin 6764

Semester Long - Project Proposal.

A Paragraph, 50 words, proposal.

Tech. Elements:
1. 3D Graphics
2. Interative
3. Presentation mode — Script features
4. Integrate Real/Live Video from File(s) And/or from a Camera

Guidelines for Topics Selection
Enhance/Improve Productive
Promote/Present Better Communication

By Sept. 23rd Submission to Email; fina.li@sjsu.edu

Show + Tell

Note: For more general cases,

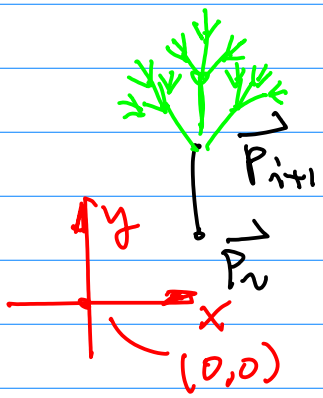


Fig.8

Sept 16 (Th)

Topics: 1° 2D Transforms
To project Implementation
Trees/Forest

Note: Homework (Show + Tell)

Homework (Un-Official)
Rotating Squares OpenGL
Reference: On git
github/fnwalli/opengl/
computer~/F2018/~
2021F-b-... Homework.

(2 pts.)

Sept. 2nd

Homework: Implementation of Rotating Squares Based on Eqn(2a)-(2b), on pp 8.

Example: On Window.

Note:
1. Always start your program with a Header;
a. Program Name:
Coded by:
Date:
Version:
Status:
Note: (Environment, O.S Compilation & Build)

2. Implementation.
   Vector Formulation w/o Rotation Matrix.

Note: SinX or CosX
Taylor Expansion (Taylor Series)

$$f(x)=f(x_0)+\frac{f'(x)}{1!}(x-x_0)+\frac{f''(x)}{2!}(x-x_0)^2+\cdots+R_n(x)$$

Look-up table.

3. Change Polygons to Lines
   2D Vertex is better for this implementation;
4. $\lambda = 0.8$ to Begin with, then when finish the entire program execution, modify λ, $\lambda = 0.95$, Observe the difference patterns.
   try $\lambda = 0.05$
5. Add Delay.
   Frame Rate (Refrash Rate for Display).
   30 FPS (Frames per Second)
   $f = 30Hz$;
   $T = 1/f = 33.3\ mS$.
6. Keep one color at time for a set of all rotating squares
7. Create multiple sets of Rotating Squares, By Defining "Anchor point" for Each set. Change
   2D Vertex (x_i, y_i)
   to
   2D Vertex (x_i+x_0, y_i+y_0)
   where my anchor point is (x_0, y_0)

Later, you can randomize the anchor point.

8. Modify the Line width

Note for the Homework, 1.3

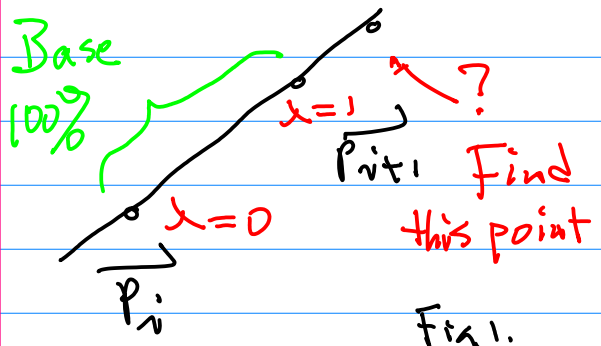


Fig1.

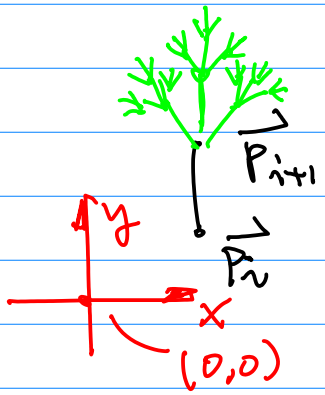


Fig3.

Now, Consider Question 2

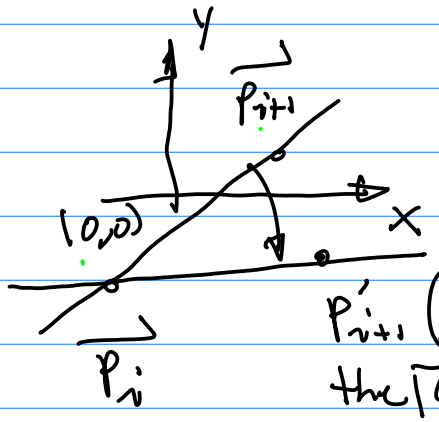


Fig2

$\vec{P}'_{i+1}$ (After the Rotation)

To Rotate the Tree Branches, we want to make sure the rotation is performed wrt the origin.

Translate $\vec{P}_{i+1}$ to the origin,

Translation matrix T to move $\vec{P}_{i+1}$ to the origin.

Pre-processing: to move $\vec{P}_i$ to the origin (0,0)

then, Rotation;

Post-processing.

$$x'_{i+1} = x_{i+1} + \Delta x \quad \text{(After)} \quad \text{(Before)} \quad \ldots (1)$$

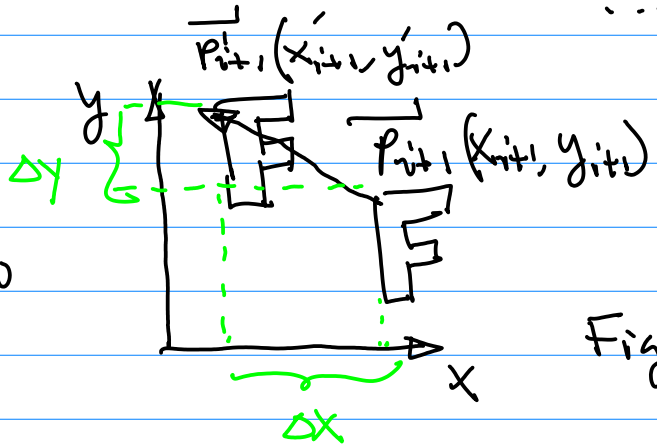


Fig4.

$$y'_{i+1} = y_{i+1} + \Delta y \quad \ldots (1b)$$

Sept. 23rd

Topic: Trees & 2D Transformation

Homework (One A week)

Composition.

Composition of 2D Transforms

$$\begin{pmatrix} x'_{i+1} \\ y'_{i+1} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i+1} \\ y_{i+1} \\ 1 \end{pmatrix} \quad \cdots (2)$$

After — Before

$\Delta x = -10$

$\Delta y = -10$

$\vec{P_1}$

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 \\ 10 \\ 1 \end{pmatrix}$$ verified.

Then, do this for $\vec{P_2}(10,20)$, find $\vec{P_2'}$

then, Rotation

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$\alpha = 30$

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 \\ 20 \\ 1 \end{pmatrix} \quad \cdots (3)$$

Then post Processing.

Post Processing to undo the Pre-processing.

$$T^{-1} = \begin{pmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \\ 0 & 0 & 1 \end{pmatrix} \quad \cdots (4)$$

Substitute the given condition, we have

$$T^{-1} = \begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{pmatrix}$$

Integrate this into the Composition matrices, we have

$T^{-1} R T =$ ← 1st Applied to the input vector

$$\begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{pmatrix} \quad \cdots (5)$$

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha \\ 0 & 0 & 1 \end{pmatrix}$$

Sept. 30 (Thu)

Today's Topics

1° Unity. 2° 3D Computer Graphics.

Note:

1° New Homework Due A week from Today

Continue the Discussion of Eqn(5).

Step1. To Take care of RT

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} \cos\alpha\cdot 1 & -\sin\alpha\cdot 1 & \Delta x\cos\alpha - \Delta y\sin\alpha \\ \sin\alpha\cdot 1 & \cos\alpha\cdot 1 & \Delta x\sin\alpha + \Delta y\cos\alpha \\ 0 & 0 & 1 \end{pmatrix}$$

Step2.

$$T^{-1}RT =$$

$$T^{-1}\begin{pmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha - \Delta x \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha - \Delta y \\ 0 & 0 & 1 \end{pmatrix} \quad \cdots (6)$$

($\sin\alpha$, $\cos\alpha$)

Hence,

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha - \Delta x \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha - \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

($\sin\alpha$, $\cos\alpha$)

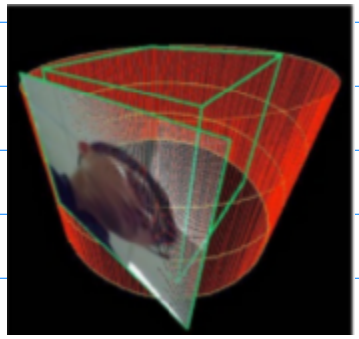
$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

Therefore,

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha - \Delta x \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha - \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

After ← → Before

e. Decoration. Texture mapping

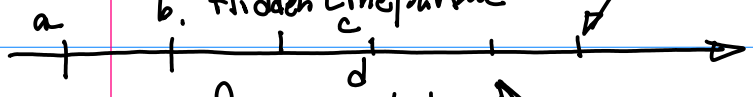


C/C++ Coding, we need

$$\begin{cases} x_i' = \cos\alpha \cdot x_i - \sin\alpha \cdot y_i + \Delta x\cos\alpha - \Delta y\sin\alpha - \Delta x \\ y_i' = \sin\alpha \cdot x_i + \cos\alpha \cdot y_i + \Delta x\sin\alpha + \Delta y\cos\alpha - \Delta y \end{cases}$$

$$\begin{cases} x_i' = \cos\alpha \cdot x_i - \sin\alpha \cdot y_i + \Delta x\cos\alpha - \Delta y\sin\alpha - \Delta x \quad \cdots (7a) \\ y_i' = \sin\alpha \cdot x_i + \cos\alpha \cdot y_i + \Delta x\sin\alpha + \Delta y\cos\alpha - \Delta y \quad \cdots (7b) \end{cases}$$

Three Dimensional Graphics. f. Lighting models.



3D World Coordinate System

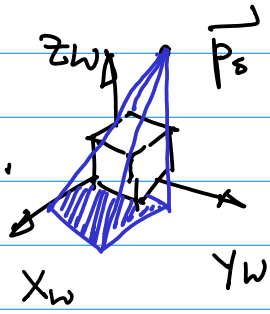
Wireframe model

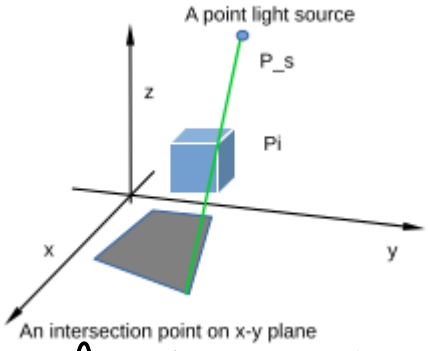
Shadows

e. Decoration Algorithm

Transformation Pipeline

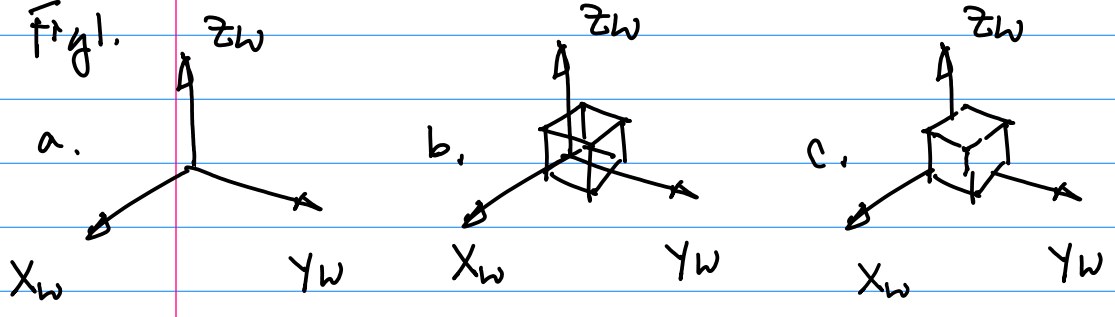
d.





from H. Li's Book manuscript.

Fig1.



f. Lighting models

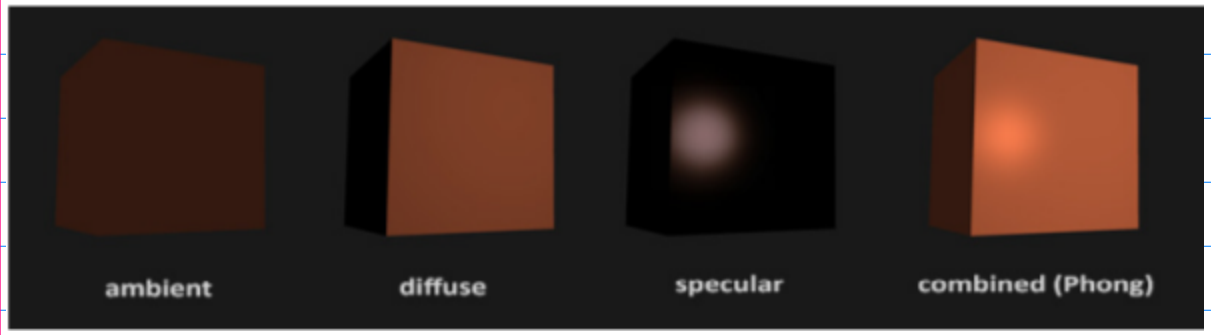


Fig.2

World Coordinate System.

Example:

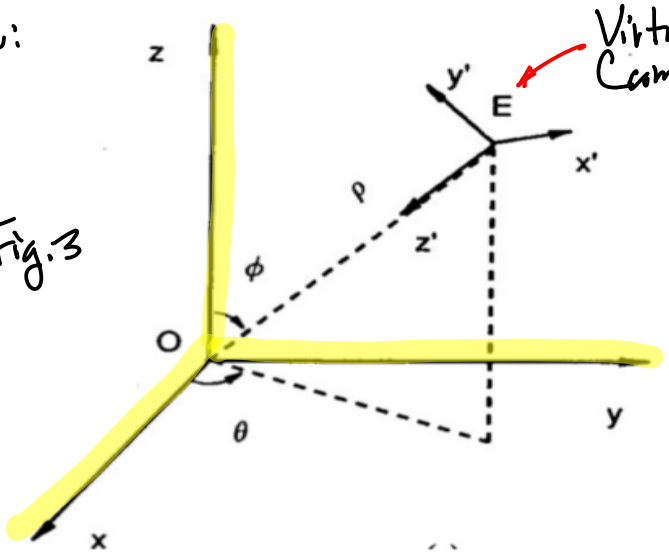


Fig.3

1° World Coordinate System (Yellow High Light)
Subscript "w" ⟶ World, x_w-y_w-z_w

2° Right-Hand System.

3° Define Everything in a World Coordinate System, including a Virtual Camera located at $E(x_e, y_e, z_e)$

4° Viewer Coordinate System, A left hand system, x_e-y_e-z_e as in Fig.4.

3D Transformation Pipelines

Step 1. Transform of a 3D object from world Coordinate System to A Viewer Coordinate System.

Step 2. Project the Object from the Viewer Transform to 2D Display Screen (Perspective Projection)

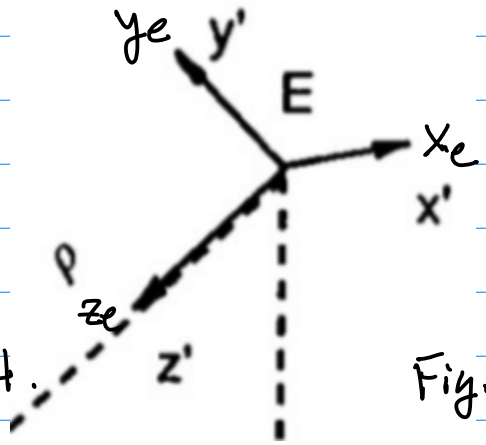


Fig.4

Fig.4

Example:

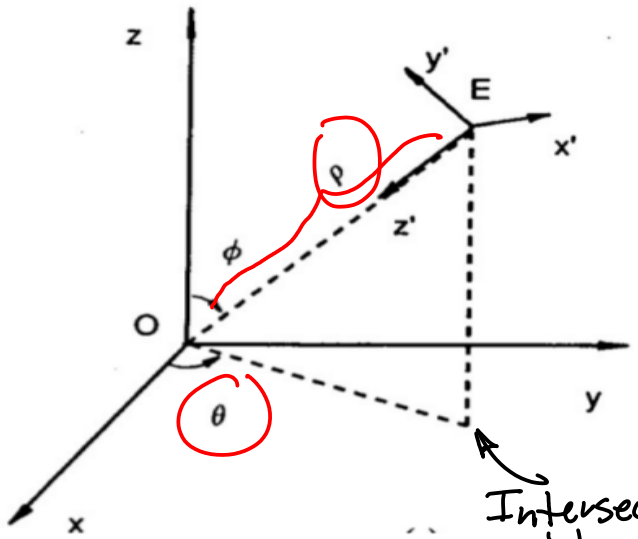


Fig5

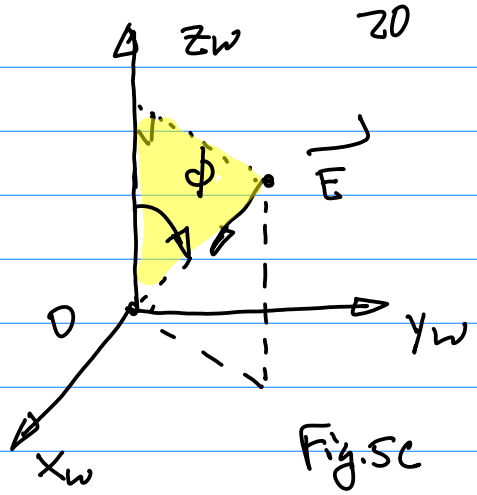


Fig.5c

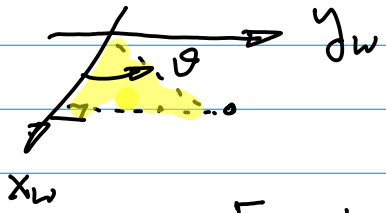


Fig5b

Intersection point

$$\mathbf{T} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{... (1)}$$

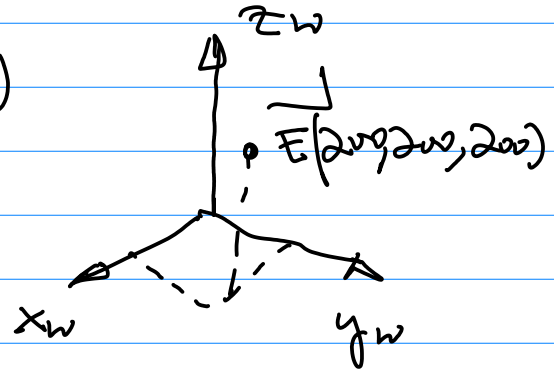
World-to-Viewer Transform

θ (Theta): Angle on X_w-y_w plane Formed by 2 vectors, 1st Vector X_w-axis, project $\vec{E}$ onto X_w-y_w plane, Link the Intersection pt on the X_w-y_w plane to the origin (0,0,0) to form the 2nd vector.

ϕ (phi): Angle Between Z_w and $\vec{E}$ Vector

ρ (rho): Distance, from $\vec{E}$ to the origin of the World Coordinate System.

Given the $\vec{E}(2w, 2w, 2w)$, find World To Viewer Transform By Eqn(1).



Sol. From Eqn(1)

We need to find θ, $\cos\theta$, $\sin\theta$; ϕ, $\sin\phi$, $\cos\phi$; ρ.

Oct. 7 (Th)

Topics: 1° 3D Transformation Pipeline

2° Homework 5 Key's Presentation of Homework 5.

Homework 6 (1pt) One / week

Submission on-line (CANVAS)

(3D X_w-Y_w-Z_w Axis Display)

Example:

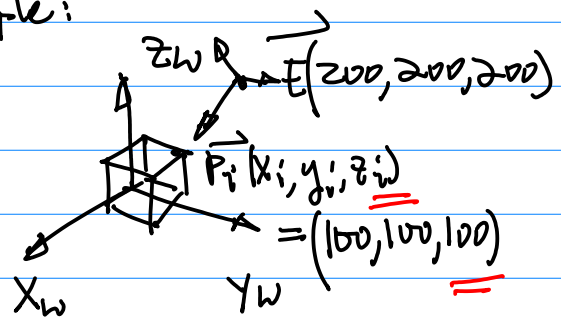


Fig. 1.

$\vec{P_i}(x_i,y_i,z_i)$ Before the Transform (in the World)

$\vec{P_i'}(x_i',y_i',z_i')$ After the Transform.

Find: 1. World-To-Viewer Transformation matrix;

2. find New point $\vec{P_i'}(x_i',y_i',z_i')$ After the Transformation.

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad \cdots (1)$$

First, find $\sin\theta$

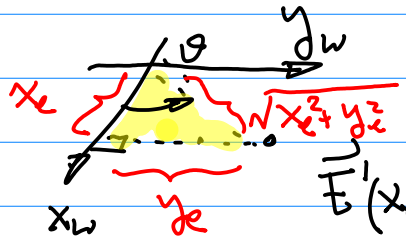


Fig. 2

$\vec{E}'(x_e,y_e,z_e)$ is formed as an intersection Point between A perpendicular line passing through $\vec{E}(x_e,y_e,z_e)$ and x_w-y_w plane

Hence, $\sin\theta = \frac{y_e}{\sqrt{x_e^2+y_e^2}} = \frac{200}{200\sqrt{2}} = \sqrt{2}/2$

$\cos\theta = \frac{x_e}{\sqrt{x_e^2+y_e^2}} = \frac{200}{200\sqrt{2}} = \sqrt{2}/2$

Similarly, find $\sin\phi$

Draw a line passing $\vec{E}$ Perpendicular to Z_w Axis to form a Triangle

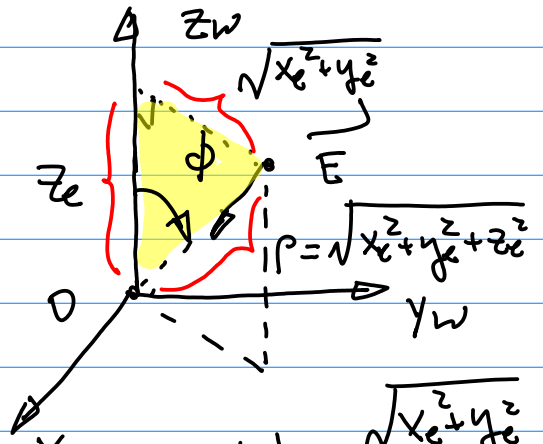


Fig3

$$\sin\phi = \frac{\sqrt{x_e^2+y_e^2}}{\sqrt{x_e^2+y_e^2+z_e^2}}$$

$$= \frac{200\sqrt{2}}{200\sqrt{3}} = \sqrt{2/3} = \sqrt{6}/3$$

$$\cos\phi = \frac{z_e}{\sqrt{x_e^2+y_e^2+z_e^2}} = \frac{200}{200\sqrt{3}}$$

$$= \sqrt{3}/3$$

$$\begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ -\sqrt{3}/3\sqrt{2}/2 & -\sqrt{3}/3\sqrt{2}/2 & \sqrt{6}/3 & 0 \\ -\sqrt{6}/3\sqrt{2}/2 & -\sqrt{6}/3\sqrt{2}/2 & -\sqrt{3}/3 & 200\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ -\sqrt{6}/6 & -\sqrt{6}/6 & \sqrt{6}/3 & 0 \\ -\sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & 200\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then, the point $\vec{P_i}$ after the Transform is $\vec{P_i'}$, given below

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \\ 1 \end{bmatrix} = \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ -\sqrt{6}/6 & -\sqrt{6}/6 & \sqrt{6}/3 & 0 \\ -\sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & 200\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

From Fig1. pp21

where $\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 1 \end{bmatrix}$ ✓

$$x_i' = -\frac{\sqrt{2}}{2}\times 100 + \frac{\sqrt{2}}{2}\times 100 = 0$$

$$y_i' = -\frac{\sqrt{6}}{6}\cdot 200 - \frac{\sqrt{6}}{6}\cdot 200 + \frac{\sqrt{6}}{3}\cdot 100$$

$$= -\frac{\sqrt{6}}{3}\times 100 + \frac{\sqrt{6}}{3}100 = 0$$

$$z_i' = -\frac{\sqrt{3}}{3}\times 200 - \frac{\sqrt{3}}{3}\times 200 - \frac{\sqrt{3}}{3}\times 200 + 200\sqrt{3}$$

$$= -200\sqrt{3} + 200\sqrt{3} = 100\sqrt{3}$$

Check physical meaning:

C/C++ Implementation. From Eqn(1)

$$\begin{aligned} x_i' &= -\sin\theta\, x_i + \cos\theta\, y_i \\ y_i' &= -\cos\phi\cos\theta\, x_i - \cos\phi\sin\theta\, y_i + \sin\phi\, z_i \\ z_i' &= -\sin\phi\cos\theta\, x_i - \sin\phi\cos\theta\, y_i \\ &\quad - \cos\phi\, z_i + \rho \end{aligned} \quad \cdots (2)$$

2nd Step of Transformation Pipeline:

Perspective Projection

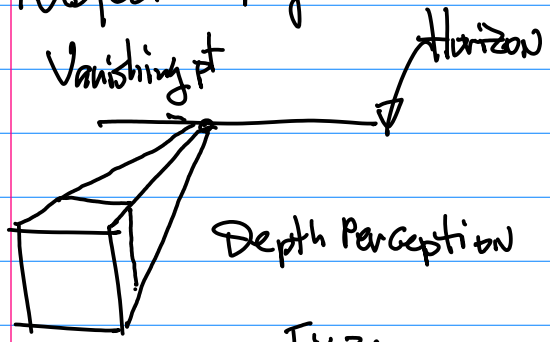
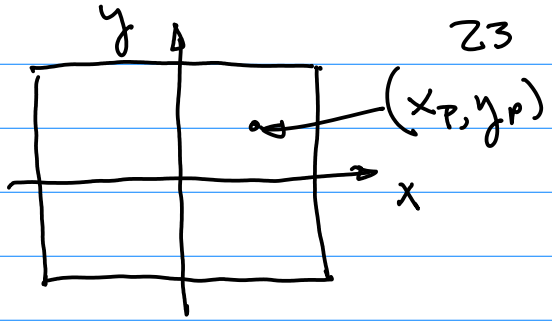


Fig3a



Example: Given a point $\vec{P}_i'$ (after World-to-Viewer Transform), Compute Perspective projection

Sol

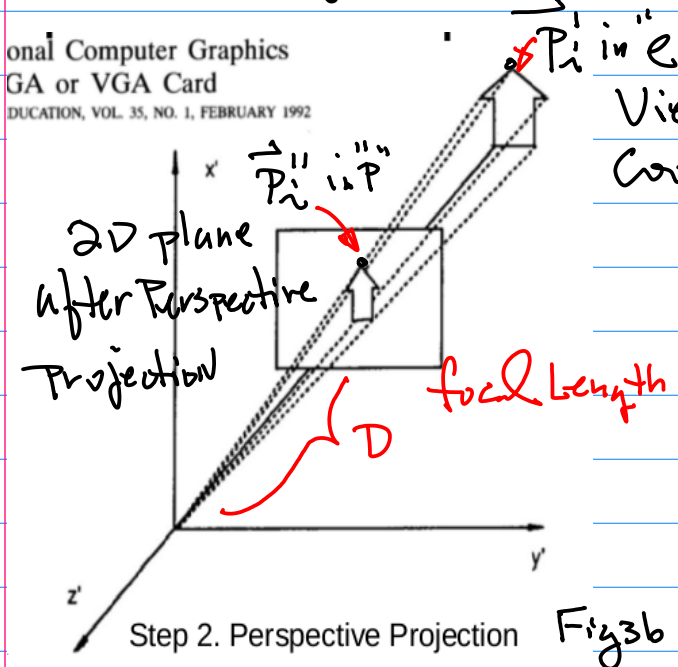
(1) Assume $D=20$ from Eqn 3).

$$x_i'' = x_i' \frac{D}{z_i'}\Big|_{D=20} = \frac{20}{z_i'} \cdot x_i'$$

$$= \frac{20}{100\sqrt{3}} x_i'\Big|_{x_i'=0} = 0$$

$$y_i'' = y_i' \frac{D}{z_i'} = \frac{20}{z_i'} y_i'$$

$$= \frac{20}{100\sqrt{3}} \cdot 0 = 0$$



Step 2. Perspective Projection Fig3b

$$x_p = x_e\left(\frac{D}{z_e}\right)$$

$$y_p = y_e\left(\frac{D}{z_e}\right)$$

... (3)

Where Subscript "e" for Viewer Coordinate System, "P" for Perspective Projection

C/C++ Implementation:

$$\begin{cases} x_i'' = x_i' D / z_i' \\ y_i'' = y_i' D / z_i' \end{cases}$$

Oct. 14 (Th)

Note: 1° Homework Submission

Migrate All Submissions to CANVAS.

(1) Put a list of all the homeworks publish it on CANVAS;

(2) Submit All your homework in one zip. to CANVAS

Due 1 week from Today.

Oct 21st.

Midterm Exam: In 2 weeks

Oct. 28th.

Today's Topics:

1° Examples of (World-To-Viewer, Perspective Projection) Transformation Pipelines.

2° Shadows

Example: Step1 for Homework3.

3. Implement world coordinate system and draw it on the screen.

Submission requirements:
(1) source code;
(2) compiled binary;
(3) 5 second video clips;
(4) put all in a zip file with the following naming convention:
FirstName-LastName-nameOfProject-4digitsID-YY-MM-DD.zip
for example:
Harry-Li-3Dworld-1116-2021-9-30.zip.

Design Data set, e.g. a collection of Vectors, $\vec{P_0}, \vec{P_1} \cdots \vec{P_i}$ to define the

World Coordinate System

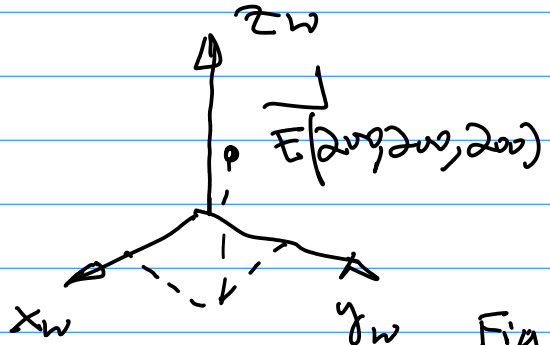


Fig. 1

$\vec{P_i}(x_i, y_i)$ to cover all axis

For 2 pts for the X-axis

Origin $\vec{P_0}(x_0, y_0, z_0) = (0, 0, 0)$

$\vec{P_1}(x_1, y_1, z_1) = (50, 0, 0)$

C/C++ for X-axis: Group $\vec{P_0}, \vec{P_1}$ together

for y_w-axis, 2 pts,

$\vec{P_0}(x_0, y_0, z_0) = (0, 0, 0)$

and $\vec{P_2}(x_2, y_2, z_2) = (0, 50, 0)$

for z_w-axis, 2 pts

$\vec{P_0}(x_0, y_0, z_0) = (0, 0, 0)$

and $\vec{P_3}(x_3, y_3, z_3) = (0, 0, 50)$

Step2. Design/Define a Virtual Camera Location

$\vec{E}(x_e, y_e, z_e) = (200, 200, 200)$

Step 3. First step of the transformation pipeline, e.g. World to Viewer Transform, From Eqn(1) pp22.

$$T_{w2v} = \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ -\sqrt{6}/6 & -\sqrt{6}/6 & \sqrt{6}/3 & 0 \\ -\sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & 2\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \quad \ldots(1)$$

Find $\sin\theta, \cos\theta, \sin\phi, \cos\phi, \rho$

From Eqn(1), we can derive 3 Equations as follows

$$x_i' = -\sin\theta\, x_i + \cos\theta \cdot y_i$$

$$\begin{cases} y_i' = -\cos\phi\cos\theta\, x_i - \cos\phi\sin\theta\, y_i + \sin\phi\, z_i \\ z_i' = -\sin\phi\cos\theta\, x_i - \sin\phi\cos\theta\, y_i - \cos\phi\, z_i + \rho \end{cases} \quad \ldots(2)$$

$$\rho = \sqrt{x_e^2 + y_e^2 + z_e^2}$$

To write the Above Equation as a C Code,

```
x_prim[i] = -sin(theta) * x[i] +
            cos(theta) * y[i] ;
```

Now, for all the vectors, $\overrightarrow{P_0}, \overrightarrow{P_1}, \ldots, \overrightarrow{P_3}$

for $\overrightarrow{P_0}$, we have

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ -\sqrt{6}/6 & -\sqrt{6}/6 & \sqrt{6}/3 & 0 \\ -\sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & 2\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$\therefore \overrightarrow{P_0}' = (0,0,0)$

for $\overrightarrow{P_1}$,

$$\begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ -\sqrt{6}/6 & -\sqrt{6}/6 & \sqrt{6}/3 & 0 \\ -\sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & 2\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 50 \\ 0 \\ 0 \\ 1 \end{bmatrix} =$$

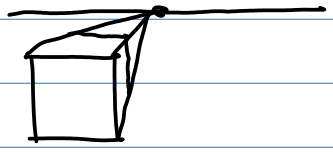
$$\begin{bmatrix} -\frac{\sqrt{2}}{2} \cdot 50 \\ -\frac{\sqrt{6}}{3} \cdot 50 \\ -\frac{\sqrt{3}}{3} \cdot 50 + 2\sqrt{3} \end{bmatrix} = \begin{bmatrix} -25\sqrt{2} \\ -\frac{25\sqrt{6}}{3} \\ -\frac{\sqrt{3}}{3} \cdot 50 + 2\sqrt{3} \end{bmatrix} \quad \ldots(3)$$

You can Compute for $\overrightarrow{P_2}$ (y_w-axis)

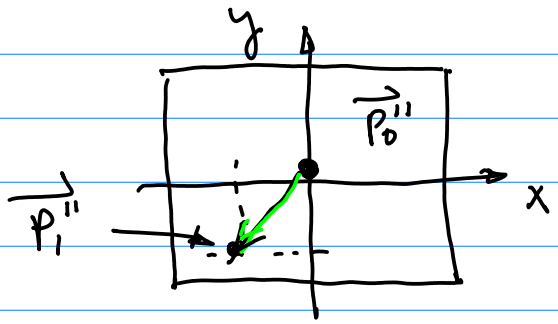
$\overrightarrow{P_3}$ (z_w-axis)

Background on Perspective Projection

Vanishing Point



Horizon



Next Step: Persective Projection

Step 4: Perspective Projection

$$\begin{cases} x_i'' = \left(\frac{D}{z_i'}\right) x_i' & \cdots(4a) \\ y_i'' = \left(\frac{D}{z_i'}\right) y_i' & \cdots(4b) \end{cases}$$

Since $\theta = 20$.

For $i=0$, $\vec{P_0}$

$$\begin{cases} x_0'' = \frac{20}{z_0'} \cdot x_0' = 0 & \cdots(5a) \\ y_0'' = \frac{20}{z_0'} \cdot y_0' = 0 & \cdots(5b) \end{cases}$$

Therefore $\vec{P_0''} = (0,0)$

For $i=1$, (x-axis)

$$x_1'' = \frac{D}{z_1'} \cdot x_1' = \frac{20}{\left(-\frac{\sqrt{3}}{3}50 + 20\sqrt{3}\right)} (-25\sqrt{2}) \quad \cdots(6a)$$

$$y_1'' = \frac{D}{z_1'} \cdot y_1' = \frac{20}{-\frac{\sqrt{3}}{3}50 + 20\sqrt{3}} \cdot \left(-25\frac{\sqrt{6}}{3}\right)$$

Similarly, for $\vec{P_2}$, $\vec{P_3}$

plot x_w-axis on 2D Display Screen. Plot the result in Green.

Sample Code for Transformation pipeline

$\vec{E}(X_e, Y_e, Z_e)$

```
28  float Xe = 100.0f;
29  float Ye = 100.0f;
30  float Ze = 100.0f;
```

https://github.com/hualili/opencv/blob/master/ComputerGraphics_AR/F2018/10-world-20190925.cpp

$$\rho = \sqrt{x_e^2 + y_e^2 + z_e}$$

```
32  float Rho = sqrt(pow(Xe,2) + pow(Ye,2) + pow(Ze,2));
33  float D_focal = 20.0f;
```

Define $\vec{P_i}(x_i, y_i, z_i)$ struct in x_w-y_w-z_w (world)

```
35  typedef struct {
36      float X[UpperBD];
37      float Y[UpperBD];
38      float Z[UpperBD];
39  } pworld;
```

```
41  typedef struct {
42      float X[UpperBD];
43      float Y[UpperBD];
44      float Z[UpperBD];
45  } pviewer;
```

Xe-Ye-Ze
Viewer Coordinate

```
47  typedef struct{
48      float X[UpperBD];
49      float Y[UpperBD];
50  } pperspective;
```

Point Struct
for Perspective Projection
2D points

Define Xw-Yw-Zw World Coordinate Below,

```
74      //define the x-y-z world coordinate
75      world.X[0] = 0.0;    world.Y[0] =  0.0;   world.Z[0] =  0.0;    // origin
76      world.X[1] = 50.0;   world.Y[1] =  0.0;   world.Z[1] =  0.0;    // x-axis
77      world.X[2] = 0.0;    world.Y[2] =  50.0;  world.Z[2] =  0.0;    // y-axis
78      world.X[3] = 0.0;    world.Y[3] =  0.0;   world.Z[3] =  50.0;   // y-axis
```

World To Viewer Transform: (3 Equation, one for Each X'_i, y'_i, z'_i)

```
184     for(int i = 0; i <= UpperBD; i++)
185     {
186         viewer.X[i] = -sPheta * world.X[i] + cPheta * world.Y[i];
187         viewer.Y[i] = -cPheta * cPhi * world.X[i]
188         - cPhi * sPheta * world.Y[i]
189         + sPhi * world.Z[i];
190         viewer.Z[i] = -sPhi * cPheta * world.X[i]
191         - sPhi * cPheta * world.Y[i]
192         -cPheta * world.Z[i] + Rho;
193     }
```

$$x_i' = -\sin\theta\, x_i + \cos\theta\, y_i$$

$$\begin{cases} y_i' = -\cos\phi\cos\theta\, x_i - \cos\phi\sin\theta\, y_i + \sin\phi\, z_i \\ z_i' = -\sin\phi\cos\theta\, x_i - \sin\phi\sin\theta\, y_i \\ \qquad\qquad - \cos\phi\, z_i + \rho \end{cases}$$

For Perspective Projection, we have

```
195     for(int i = 0; i <= UpperBD; i++)
196     {
197         perspective.X[i] = D_focal * viewer.X[i] / viewer.Z[i] ;
198         perspective.Y[i] = D_focal * viewer.Y[i] / viewer.Z[i] ;
```

$$\begin{cases} x_i'' = \left(\frac{D}{z_i'}\right) x_i' \quad \cdots (4a) \\ y_i'' = \left(\frac{D}{z_i'}\right) y_i' \quad \cdots (4b) \end{cases}$$