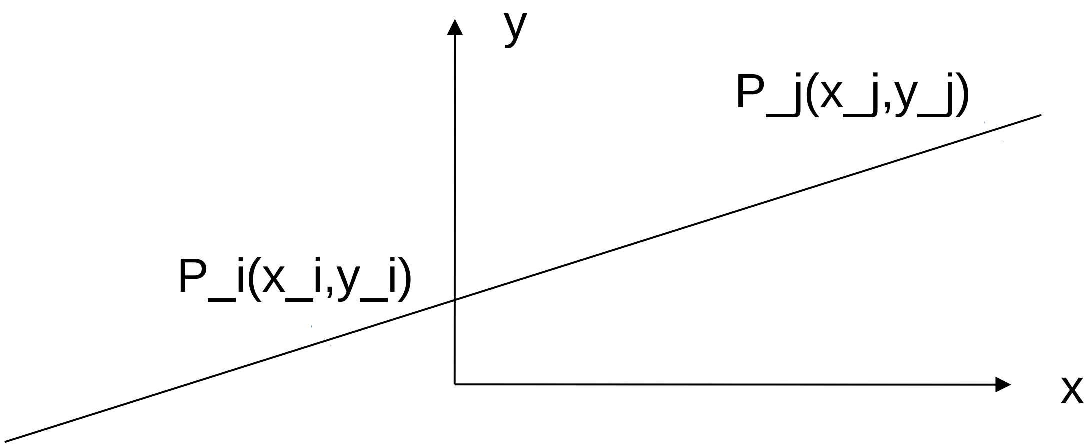
Introduction to 2D Vector Graphics



Note:
(1) for alpha = 0 and 1 equation (2) returns P(x,y) = P_i(x_i, y_i), P(x,y) = P_j (x_j , y_j);

Based on the above vector form equation, generate screen savers and trees. (Reference: H. Li, IEEE Transactions on Education)

Direction vector d(x,y) = (dx, dy), which is defined as

$$d(x,y) = P_i(x_i,y_i) - P_j(x_j,y_j) \quad \ldots (1)$$

$$= (x_i - x_j,\ y_i - y_j) \quad \ldots (1\text{-}1)$$

Now the vector form equation for the straight line:

$$P(x,y) = P_i(x_i,y_i) + alpha * (P_i(x_i,y_i) - P_j(x_j,y_j)) \quad \ldots (2)$$

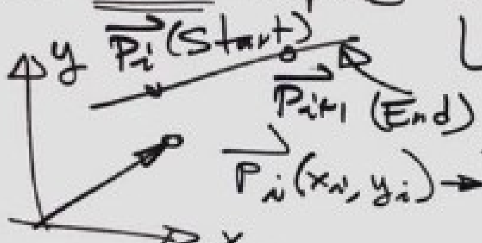
Where $0 <= alpha <= 1$

Harry Li, Ph.D.

9-17-2018 2D Vector Algorithm Example

CMPE240 Adv. Microprocessor Systems. Sept. 17, 2018 HL .3/.

2D Vector Graphics V.S. Pixel graphics



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

x-y Coordinate System.

Example: Given $\vec{P_i}$, and $\vec{P_{i+1}}$, find A direction vector $\vec{d}$.

Sol from Eqn (2), we have

$$\vec{d} = \vec{P_{i+1}} - \vec{P_i}$$

$$(x_d, y_d) = (x_{i+1}, y_{i+1}) - (x_i, y_i)$$

$$= (x_{i+1} - x_i, y_{i+1} - y_i)$$

```
x_d = x[i+1] - x[i];
y_d = y[i+1] - y[i];
```

Line Segment. $\vec{P}(x,y) \overset{?}{=} \vec{P_i} + \lambda \vec{d}$ ... (1)

where $\vec{d} \triangleq \vec{P_{i+1}} - \vec{P_i}$ ... (2)

C/C++ To Compute Eqn (2).

physical meaning of Eqn (1):

Question 1: To Get "Start" pt on the Line Segment, what is the λ=? $\lambda = 0$

To the "Ending" pt, λ=? $\lambda = 1$.

all the pts B/W $\vec{P_i}$ and $\vec{P_{i+1}}$, λ=? $0 < \lambda < 1$

all the pts Beyond $\vec{P_{i+1}}$, λ=? $\lambda > 1$

" pts " Below $\vec{P_i}$, λ=? $\lambda < 0$

9-17-2018 2D Vector Rotating Sqrs

Example: Implement 2D Vector Graphics
ON S.P.I. LCD Display.

Sol $\left\{ \overrightarrow{P_i(x_i, y_i)} \,\middle|\, i = 0, 1, 2, 3 \right\}$

From Eq(1), we have

$$\overrightarrow{P_i^{j+1}} = \overrightarrow{P_i} + \lambda\left(\overrightarrow{P_{i+1}^{j}} - \overrightarrow{P_i^{j}}\right)$$

Let $\lambda = 0.2$

Harry Li, Ph.D.

10-1-2018 2D Trees Algorithm

Example: 2nd Half Project I (2D) Trees

Step1. Initial Position (x_0, y_0) (Virtual Coordinate, GE.) $\vec{P}_1$ (x_1, y_1)

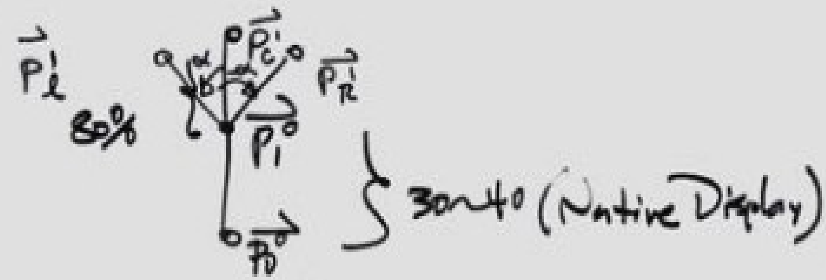
Step2. Draw Tree Trunk $\vec{P}_0$ (Width—thicker, Color: Brown)

DrawLine function to plot the Trunk; level "0", Adequate "Head Room" to grow the tree; (Level ≥ 9)

Step3 Generate 3 Child Nodes (Center, Left, Right)

Center Node: Same Direction as its previous Parent Node w/80%, $\vec{P}^1_c$

left Node $\vec{P}^1_l$ w/ $\alpha = 30°$ (Counterclockwise, "+"), Right Node $\vec{P}^1_R$



Right Rotation (Clockwise, "−")

Step4 Repeat this Process (e.g.) go to Step3 and Continues to the Next level $j+1$ till level ≥ 7 then stop.

Computation of Center Node, $\boxed{\vec{P}_c^{j+1} = \vec{P}^j_{c,L,R} + \lambda \vec{d}_{c,L,R}}$

$$(x_c^{j+1}, y_c^{j+1}) = (x^j_{c,L,R}, y^j_{c,L,R}) + 0.8(x^j_{d/c,L,R}, y^j_{d/c,L,R})$$

$$\begin{cases} x_c^{j+1} = x^j_{c,L,R} + 0.8\, x^j_{d/c,L,R} = x^j_{c,L,R} + 0.8(x^j_{c,L,R} - x^{j-1}_{c,L,R}) \\ y_c^{j+1} = y^j_{c,L,R} + 0.8\, y^j_{d/c,L,R} \end{cases} \quad \cdots (1)$$

Harry Li, Ph.D.

10-1-2018 Rotation And Translation

CMPE240 Oct. 1st, 2018 Harry Li 2/.

Rotation Matrix.

$$\underset{\text{After}}{\begin{pmatrix} x_1' \\ y_1' \\ 1 \end{pmatrix}} = \underbrace{\begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_{3\times3}}{}^{3\times3} \underset{\text{Before}}{\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}} \quad \cdots(2)$$

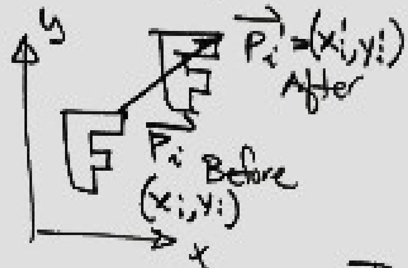
$$\begin{cases} x_1' = \cos\alpha\, x_1 - \sin\alpha\, y_1 ; & \cdots(2^*) \\ y_1' = \sin\alpha\, x_1 + \cos\alpha\, y_1 ; & \end{cases}$$

Regarding 2nd Type of Rotation.

1. Move the Reference pt to the origin (0,0) — Preprocessing Stage
2. Rotate By Eqn (2)
3. Move Back to its original Location — Post Processing.

Preprocessing.

$$\underset{\text{After}}{\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix}} = \underbrace{\begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{"}T_{3\times3}\text{ Translation"}} \underset{\text{Before}}{\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}} \quad \cdots(3)$$



$$\begin{cases} x_i' = x_i + \Delta x ; \\ y_i' = y_i + \Delta y ; \end{cases} \quad \cdots(3^*)$$

Post Processing:

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \\ 0 & 0 & 1 \end{pmatrix}}_{T^{-1}_{3\times3}}{}^{3\times3} \begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} \quad \cdots(4)$$

10-1-2018 3 Steps For Arbitrary Rotations

Now, Consider Put All 3 OPS together

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} \overset{?}{=} \underbrace{(T^{-1})_{3\times3}(R)_{3\times3}(T)_{3\times3}}_{\text{C/C++ Code}} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

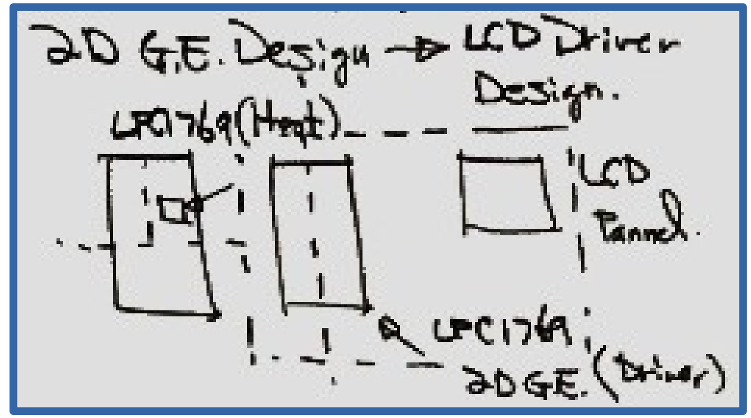
After   C/C++ Code   Before

Programming Aspect:

$$T_{\Sigma} = T^{-1}_{3\times3} \cdot R_{3\times3} \cdot T_{3\times3}$$

$\downarrow$

2 Lines of C/C++



Harry Li, Ph.D.

2D Vector Graphics to Create Rotating Pattern

P^(k-1)_i(x_i,y_i)

P^k _i(x_i,y_i)

P^k _i+1 (x_i+1,y_i+1)

P^(k-1)_i+1(x_i+1,y_i+1)

P^(k-1)_i+3(x_i+3,y_i+3)

P^k _i+2 (x_i+2,y_i+2)

P^(k-1)_i+2(x_i+2,y_i+2)

P^k _i+3 (x_i+3,y_i+3)

From equation (1), we can derive the following equation

P^k (x,y) =   P^(k-1) _i(x_i,y_i) + alpha * (P^(k-1)_i(x_i,y_i) – P^(k-1)_j(x_j,y_j)) … (3)

Choose alpha = 0.8

Harry Li, Ph.D.

2D Rotating Pattern Technique

I have created an algorithm to summarize the rotating pattern technique:

Defining a polygon with a set of vertices $\{(x_i, y_i) | i = 1, 2, \cdots, k\}$, one can use a vector formula to describe an object reduction and rotation as:

$$\left(x_i^{l+1}, y_i^{l+1}\right) = \left(x_i^l, y_i^l\right) + \mu\left(x_{i+1}^l - x_i^l, y_{i+1}^l - y_i^l\right) \qquad (1)$$

where the subscript i is used to denote each vertex of a given object, $i = 1, 2, \cdots, k$. If $i = k$ and $i + 1 > k$, then $i + 1 = 1$. The superscript l is used to denote the level of iteration. The constant μ is defined as $0 \leq \mu \leq 1$, which is used to define the rate of reduction. For example, if $\mu = 0.5$ then each side of the object is reduced to half. This μ is also related to the direction of the rotation. For μ less than 0.5, the rotation is toward the current reference point $\left(x_i^l, y_i^l\right)$, otherwise the rotation is away from the current point. Equation (1) is, in fact, derived directly from a vector addition. Let us assume $\bar{a} = \left(x_i^l, y_i^l\right)$, $\bar{b} = \left(x_{i+1}^l - x_i^l, y_{i+1}^l - y_i^l\right)$, and $\bar{a}' = \left(x_i^{l+1}, y_i^{l+1}\right)$, then (1) can be written as $\bar{a}' = \bar{a} + \mu\left(\bar{b} - \bar{a}\right)$ for vertices i and $i + 1$ with reduction at level l. Following the argument above, μ can also be expanded to the range greater than 1 to produce magnification.

Reference:

IEEE TRANSACTIONS ON EDUCATION, VOL. 35, NO. 1,
Three-Dimensional Computer Graphics Using EGA or VGA Card By H. Li