Smart

Team

ملخص جرافيكس

بإعداد : ميرال زريقات

السعر

للتواصل معنا

تطبيق سمارت
سمارت تيم

يوتيوب
SmartteamO16

فيس بوك
Group/Smart Team

انستغرام
SmartteamO16

Final 1/16/18

Revision date 1/4/18

Good luck ☺ Adam ☺

Graphics revision notes

Ch. 3

Primitives

1. Lines

general line equation

$y = mx + b$

where m is the slope of the line

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

There are many algorithms to ~~[illegible]~~ drawing lines

a. DDA or as it's known as Digital Differential Analyzer

case 1: where $m > 1$ $\quad y_k = y_0 + 1$

$x_k = x_0 + 1/m$

case 2: where $m <= 1$ $\quad y_k = y_0 + m$

$x_k = x_0 + 1$

b. Bresenham's Line Algorithm

for $m < 1$

$p_0 = 2\Delta y - \Delta x$

if $p_k < 0$ $\quad x_k = x_0 + 1$

$y_k = y_k$

$p_k = p_0 + 2\Delta y$

performed $\Delta x - 1$ times

if $p_k > 0$ $\quad x_k = x_0 + 1$

$y_k = y_0 + 1$

$p_k = p_0 + 2\Delta y - 2\Delta x$

Circle drawing Algorithm

Midpoint Circle Algorithm

- input radius is usually (0, r) r meaning radius of the circle
- $P_0 = 5/4 - r$ rounds up to 1 so
  $P_0 = 1 - r$
- if P_k is negative

  $P_{k+1} = P_k + 2x_{k+1} + 1$ & the next point $(x+1, y)$

- if P_k is positive the next point is $(x+1, y-1)$

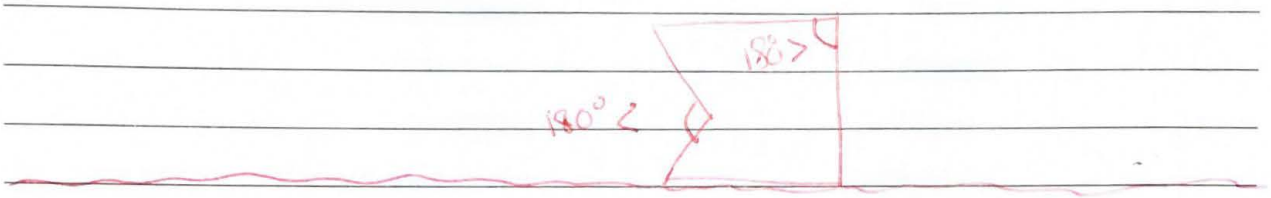
  $P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_{k+1}$

- Determine symmetry points to each other 7 octants
  يعني بدي النقطة المقابلة لكل نقطة بلاقيها في اول ثمن ع داير لي
  لل 7 الي ضلوا
- move each centred pixel ~~posisto~~ position onto a circular path
  يعني حرك كل نقطة من شكل دائري بالنسبة للمركز تاع الدائرة
  where $x = x + x_c$
  & $y = y + y_c$
- repeat the previous steps until $x >= y$.

Polygons

all that we need to learn about polygons is that there are 2 types
concave :- meaning all angles are less than 180° except for one
convex :- meaning all angles are less than 180°

* you must know that the least number of lines needed to draw a concave polygon is 5



2D geometric Transformations

- Translation :- is moving a pixel from one place to another either ~~x~~ according to the x or y axis's

where the new x is $x + t_x$ → translation in x

& the new y is $y + t_y$ → translation in y

matrix

$$\begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t_x + x \\ t_y + y \end{bmatrix}$$

T → amount of translation

P → Pixel

P' → new Pixel

- Rotation :- moving the polygon according to a specific angle

where the new x is $x\cos\theta - y\sin\theta$

By is $x\sin\theta + y\cos\theta$

$P' = P \cdot R$

Matrix

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

- Scaling :- making the polygon bigger or smaller about a fixed point or the origin

where the new x is $x \cdot Sx \rightarrow$ Scaling factor in x

By is $y \cdot Sy \rightarrow$ Scaling factor in y

$P' = P \cdot S$

$$\begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Homogeneous Coordinates

It's performing many transformations on polygons — but every Matrix had to be changed into 3*3 Matrices as it was very difficult to multiply the previous matrices —

- Translation

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

- Rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

* Remember!!
when multiplying Matrices you multiply from Right to left ←

Let's say you want to translate then Rotate

R T M

General 2D Rotation around any point

1. you translate the polygon back to the origin
2. Rotate around given angle
3. translate the polygon back to it's original place

$T \quad R \quad I^{-1} \quad M \rightarrow$ given points

General 2D Scaling around any point

1. Translate to origin
2. Scale around given factors
3. Translate back to position

$T \quad S \quad I^{-1} \quad M \rightarrow$ given points

- Reflection is, Scaling
  "considered"

around x axis

$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & -Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

but you don't want to change the size of your polygon so —

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

around y axis

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

around origin

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

around $y = x$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

around $y = -x$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

• Shearing = Distortion of the shape of the polygon

x-directed

$x' = x + sh_x \cdot y$

$y' = y$

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

y-directed

$x' = x$

$y' = y + sh_y \cdot x$

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3D Geometric Transformations

Vector - is an entity that has length & a direction, but the length doesn't matter here, the direction matters more, and it doesn't have a fixed location, it is the space between 2 points, its coordinates are calculated by substracting both points & its direction is from the nearest to farthest point

i.e:-

$V = P2 - P1$

$$\begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}$$

• Note that the only operations that vectors allow you to do are addition & multiplying them by a real number

$\{0, 1, 2, 3, -1, -2, -3, -\}$

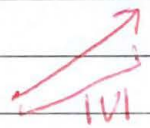
* Please excuse my handwriting — I'm sick

Vector addition

- addition means both substraction & summation, it ALWAYS gives out a new vector with the direction of both vectors or the largest, — take notes here

Normalize a vector

- Magnitude (length): - distance from tail to head



it is calculated along this formula

$$|V| = \sqrt{V_x^2 + V_y^2 + V_z^2}$$

x y z

coordinates

normalization is turning any vector to a unit vector

$$\hat{V} = \frac{V}{|V|}$$

$\hat{V}$ ← unit vector

V ← vector coordinates (x, y, z)

$|V|$ ← vector length (magnitude)

• Multiplication in vectors is not like normal multiplication due to an angle ($\theta, \phi, \alpha, \beta$) between the 2 vectors or more wanting to be multiplied, in physics, the 2 ways to multiply vectors is by

a. Dot product ($\cdot$)
according to the following formulas

1. $V1 \cdot V2 = |V1| * |V2| \cdot \cos\theta$ ← cosine of the angle
(|V1| * |V2|: length of vectors)

2. $V1 \cdot V2 = V_{1x} V_{2x} + V_{1y} V_{2y} + V_{1z} V_{2z}$

* Sometimes it is asked to find the cosine of the angle it is calculated according to the following formula

$$\cos\theta = \frac{|V1||V2|}{V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z}}$$

b. Cross Product (X)
according to the following formula

$V1 \times V2 = u|V1||V2| \sin\theta$
where u is a unit vector ($\hat{x}, \hat{y}, \hat{z}$) perpendicular to V1 & V2

$$V1 \times V2 = \begin{bmatrix} u_x & u_y & u_z \\ V_{1x} & V_{1y} & V_{1z} \\ V_{2x} & V_{2y} & V_{2z} \end{bmatrix} \Rightarrow \begin{bmatrix} V_{1y}V_{2z} - V_{2y}V_{1z} \\ V_{1z}V_{2x} - V_{1x}V_{2z} \\ V_{1x}V_{2y} - V_{1y}V_{2x} \end{bmatrix}$$

- Translation

in 3D the 3*3 Matrix is replaced by a 4*4 Matrix

$$\begin{bmatrix} 1 & 0 & 0 & \delta x \\ 0 & 1 & 0 & \delta y \\ 0 & 0 & 1 & \delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (x+\delta x) \\ (y+\delta y) \\ (z+\delta z) \\ 1 \end{bmatrix}$$

- Rotation

NOW! ⇒ everyone has trouble with this but it's pretty simple

⇒ when you choose an axis to rotate the shape around that axis' coordinates don't change

now 2D rotation is basically rotating around the z-axis

so :-

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta - y\cos\theta$$
$$z' = z$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

around x-axis

$$x' = x$$
$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

around y-axis

$$y' = y$$
$$x' = z\sin\theta + x\cos\theta$$
$$z' = z\cos\theta - x\sin\theta$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

* General 3D Rotation

هون بحكي انو بدو يعمل Rotation حوليه كل الـ axis هلأ الـ 2D Rotation بحكي الـ cartesian axis xyz فالمعادلات زي

بس الفرق هون انو 3D فلازم يكون عندك زاويتين β, α او ϕ, θ

Steps

1. Translate the object back to origin
2. Rotate the object around an axis that will eventually coincide with a cartesian axis.
3. Rotate around cartesian axis
4. Rotate
5. inverse Rotate around rotation axis to return to previous direction
6. Translate the object back to its original position

3D Scaling

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

* around a fixed point

1. Translate back to origin
2. Scale around the origin
3. Translate back into the fixed point

*Chapter 6 2D viewing

Switching from World coordinates into device coordinates or viewing coordinates

We have 2 commands in OpenGL

- glOrtho2D(xwmin, xwmax, ywmin, ywmax)
  to set the clipping window coordinates
- glViewport(xvmin, yvmin, xvmax, yvmax)
  to set viewport coordinates ← what you want to show from your model

2D viewing ~~pipe~~ pipeline :-

Modeling Coordinates (MC) → World Coordinates (WC) → Viewing Coordinates (VC) Normalized Viewing Coordinates (NVC).

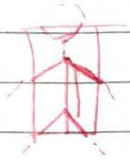
How to do that :-

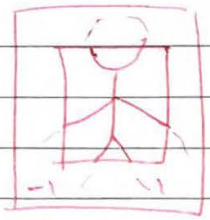
- Construct the word coordinates using MC transformations
  ↳ translation / rotation / scaling
- convert WC → VC clipping what you want to show from your model
- Transform your viewing coordinates to normalized (usually 0 → 1 or −1 → 1)
- last thing is to transform NVC → DC
  Meaning to "project" your model on your screen

Example

you want to draw a stick figure

1)  Model

2)  transforming into WC

3) 

NVC in ViewPort

In 3D we project our Model using a camera

- Orthographic Projection

It is parallel to the z axis always & perpendicular to the projection plane

$P_{ortho}(x, y, n)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Perspective projection

$P_p \left(\frac{x}{z/n}, \frac{y}{z/n}, n\right)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

- Illumination

Here it's just showing details of your model

by

lighting & texuring ← surface rendering

↓

shading

↓

Lighting the model

- Lighting : in our everyday life objects either produce or reflect light in coding we have 2 models that emitt (produce) light

a. Distributed (area) light source

produces a ray of light (ضوء الشمس)

b. Point light source

(ضوء spotlights)

Illumination

- ~~[illegible]~~ Models are methods to calculate light intensity
  عشان نحسب شدة الإضاءة اللي رح تبين الـ Model تاعنا

* هون بنشبه الفيزياء ، انو رح نكون نطبق ع قوانين الرؤية و
الضوء في الفيزياء

~~[illegible]~~ Models have :- position
Illumination
- Orientation
- Material → what is the material we are pointing the light at
- Light source
- Viewer

- We have 2 types of illumination models
  a. local :- concerns the interchange of light sources
  بين اختلاف مصدر الضوء
  b. general :- concerns the interchange of light between surfaces يعني كيف بينتقل الضوء من زجاج ع خشب و هيك
    - Ray-Tracing, light as particle — انو نتتبع الضوء
    - Radiosity, light as energy — اما ع شكل جزيئات او ك
ع شكل طاقة لانو بالفيزياء الضوء بعتبر الاشي 6 طاقة و جزيئات

- Phong Model
  - A model created in 1975 to settle between acceptable results and processing costs

- its concept is that when a light interacts with a solid object it gets reflected, absorbed, scattered & transmitted.

  Light at surface = absorbed + scattered + reflected + transmitted

- it shows reflected light as

  reflected = ambient + diffuse + specular

  ambient: الضوء المحيط عبر ال model او الكون
  diffuse: الضوء المنتشر من ال model
  specular: الضوء الذي ارتد من الجسم

Matrices for final

- 3D viewing

orthogonal projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

perspective projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

where n is near

- 2D viewing

in 2D viewing we scale and translate according to the following

$$S = \begin{bmatrix} s_x & 0 & x_{wmin}(1-s_x) \\ 0 & s_y & y_{wmin}(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

$x_{wmin}(1-s_x)$ → world x coordinates

$y_{wmin}(1-s_y)$ → world y coordinates

$$T = \begin{bmatrix} 1 & 0 & x_{vmin} - x_{wmin} \\ 0 & 1 & y_{vmin} - y_{wmin} \\ 0 & 0 & 1 \end{bmatrix}$$

$x_{vmin} - x_{wmin}$ → viewports x coordinates

$y_{vmin} - y_{wmin}$ → viewport y coordinates

- for lighting the equation & all its parts are practically given

$I_a, I_p, k_a, k_d, O_{d_x}, f_{att}, d_L$, ~ etc

keep in mind the dot & cross products of vectors

↓ we use this in illumination mostly

↓ usually is there when he wants us to write a code

- dot product

$V_1 \cdot V_2 = V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z}$ ← if not use this.

or!

$|V_1||V_2| \sin\theta$ ← if given θ use this

- cross product is the more "complicated"

$$V_1 \times V_2 = \begin{bmatrix} V_{1y}V_{2z} - V_{1z}V_{2y} \\ V_{1z}V_{2x} - V_{1x}V_{2z} \\ V_{1x}V_{2y} - V_{1y}V_{2x} \end{bmatrix}$$

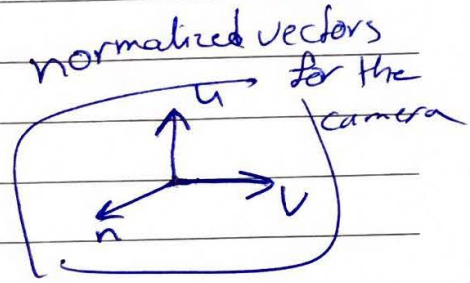
- Normalizing your vector

$|V|$: length of vector → $\hat{V} = \frac{V}{|V|}$ (x, y, z)

$|V| \quad \sqrt{V_x^2 + V_y^2 + V_z^2}$

- Camera's position Matrix

$$\begin{bmatrix} u_x & u_y & u_z & -u \cdot p_0 \\ v_x & v_y & v_z & -v \cdot p_0 \\ n_x & n_y & n_z & -n \cdot p_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



you are usually given u, v, n

* note that this matrix is after translating & rotating

- Reflection of light

$$R + L = (2N \cdot L)\,N$$
$$R = (2N \cdot L)\,N - L$$