FDF
(FILS DE FER)

3D Wireframe

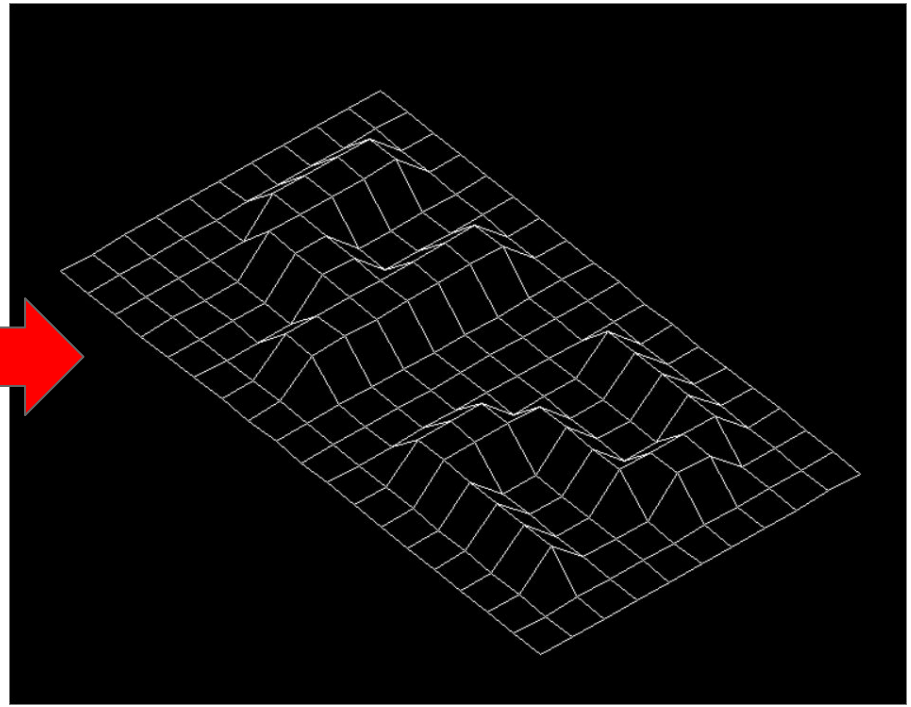
Index

Objective

Display text with numbers

Separated by integers in wiregrid

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0 10 10  0  0 10 10  0  0  0 10 10 10 10 10  0  0  0
 0  0 10 10  0  0 10 10  0  0  0  0  0  0  0 10 10  0  0
 0  0 10 10  0  0 10 10  0  0  0  0  0  0  0 10 10  0  0
 0  0 10 10 10 10 10 10  0  0  0  0 10 10 10 10  0  0  0
 0  0  0 10 10 10 10 10  0  0  0 10 10  0  0  0  0  0  0
 0  0  0  0  0  0 10 10  0  0  0 10 10  0  0  0  0  0  0
 0  0  0  0  0  0 10 10  0  0  0 10 10 10 10 10 10  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```



Mandatory & Bonuses

Mandatory - translate user input to wire grid

Bonus -

(i) rotation

(ii) different projections

(iii) zooming

(iv) translation

(Highly suggest implementing bonuses if tackling this project!)

Key learnings

Level of Abstraction

MiniLibX function/tools

- Only able to place pixel by pixel

Standard inputs when using minilibX

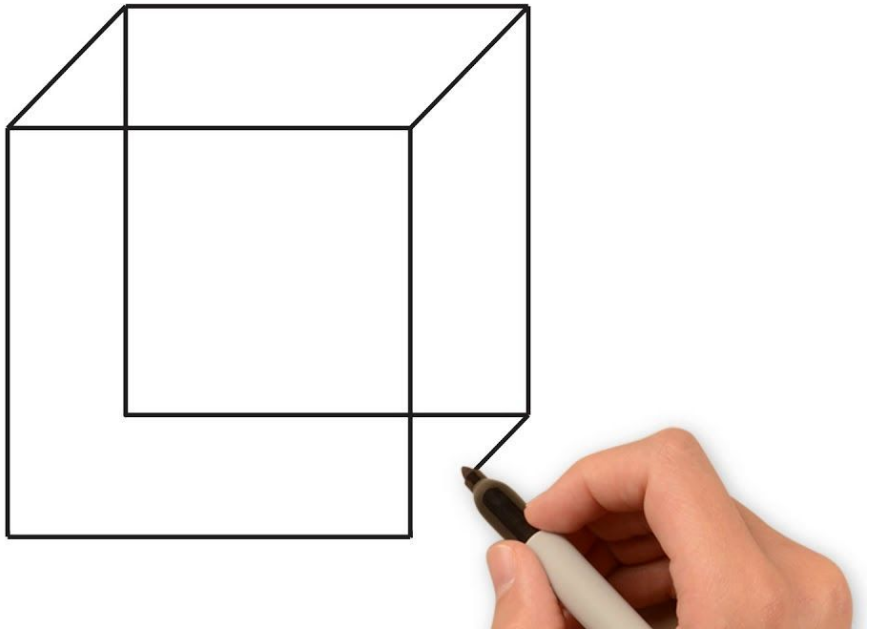
```
mlx_pixel_put(mlx, window, 50, 50, 0x0000FF00);
```

x, y coordinates

Color in hexadecimal

Level of Abstraction

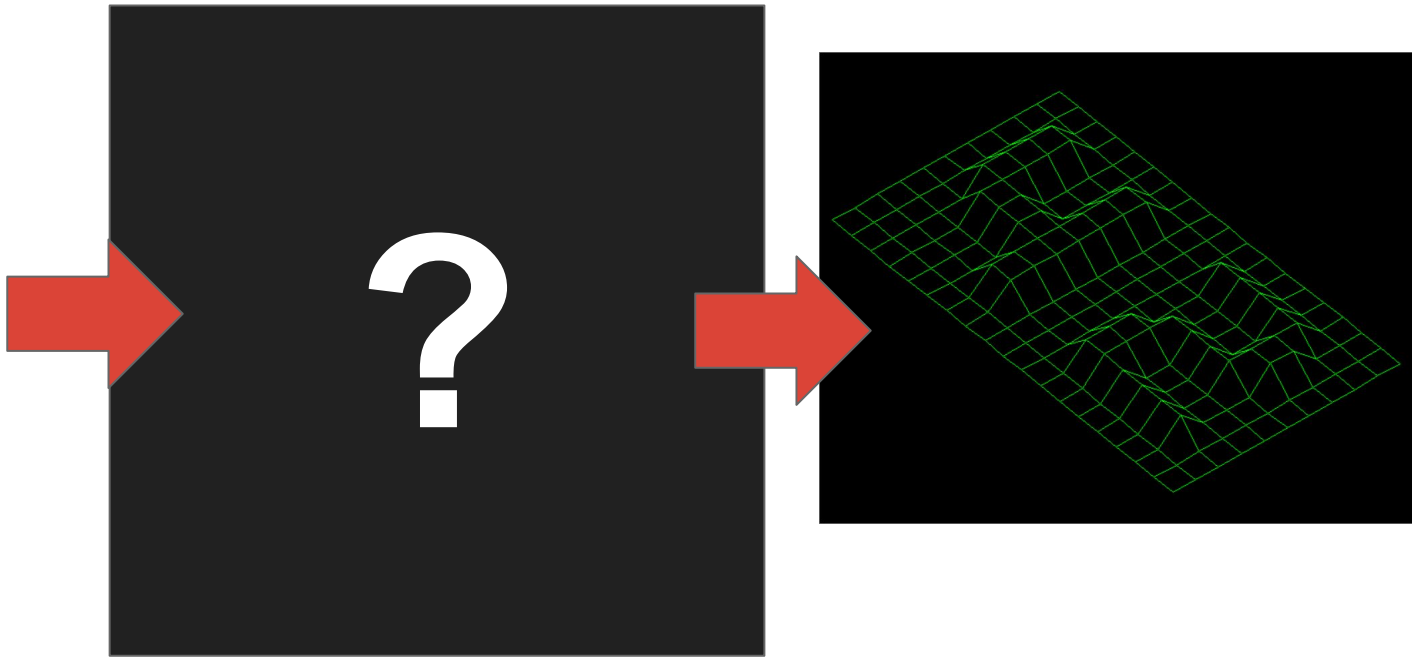
Imagine drawing a cube on a piece of paper, but instead we're 'drawing' **pixel by pixel**.



Level of Abstraction (cont'd)

MiniLibX function/tools

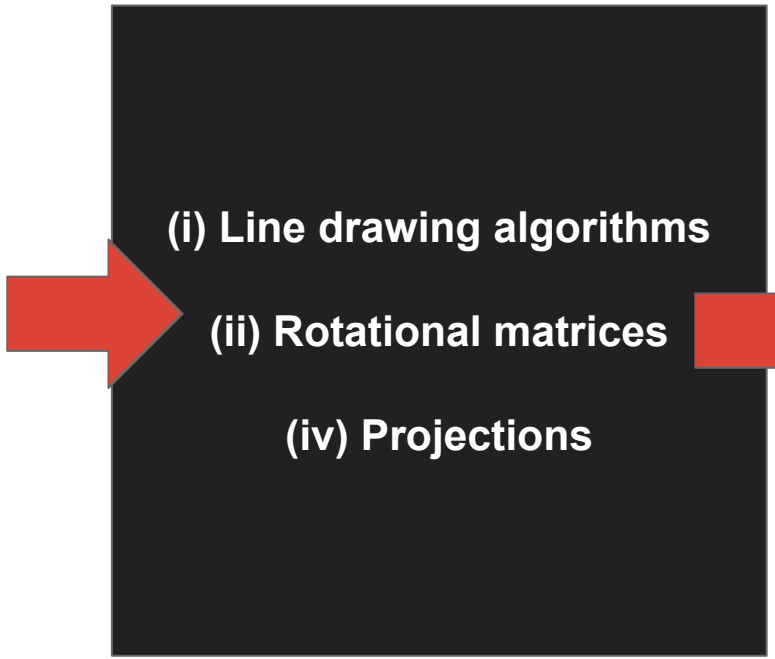
- Only able to place pixel by pixel

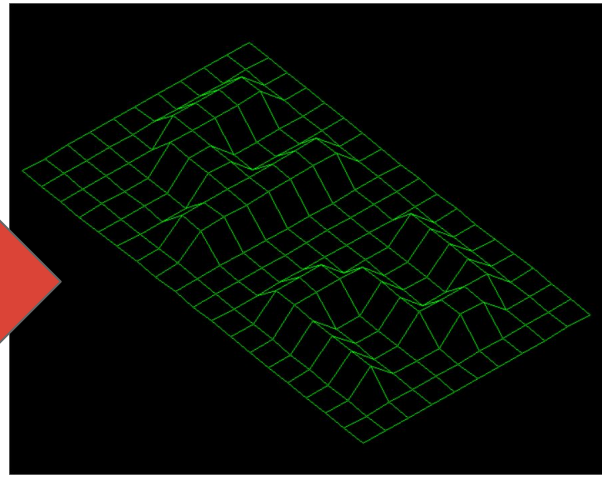


Level of Abstraction (cont'd)

MiniLibX function/tools

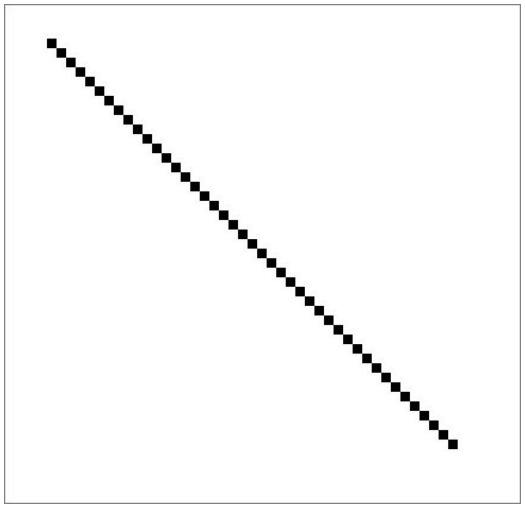
- Only able to place pixel by pixel

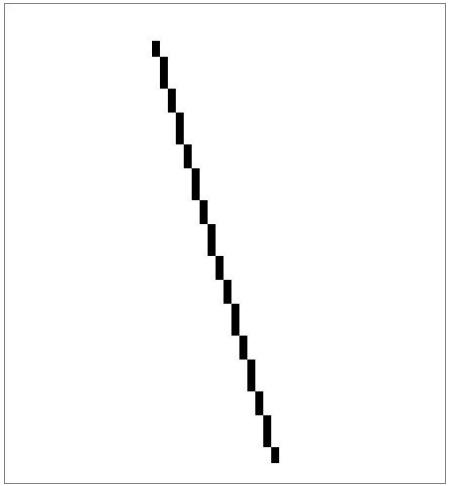


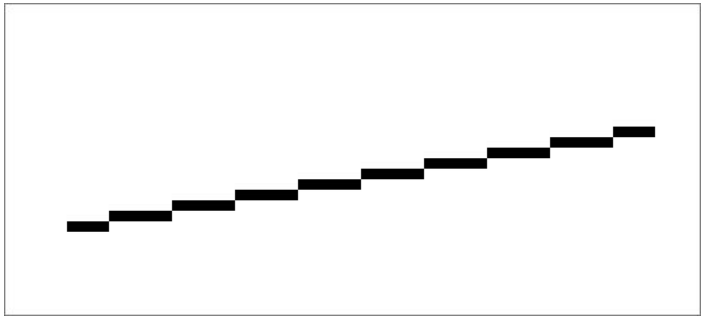


Line Drawing Algorithms

Since we are placing pixel by pixel, how do we write a function that can draw these lines?



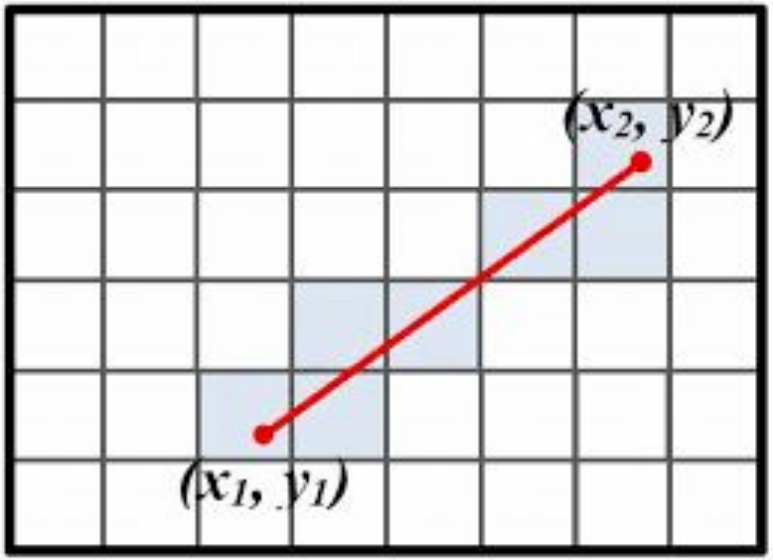




Line Drawing Algorithms (cont'd)

Issue:

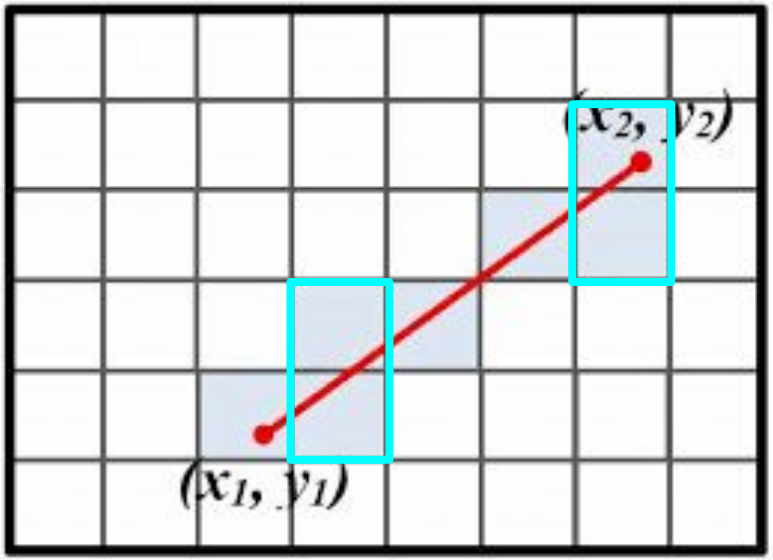
How to proportionately draw pixels?



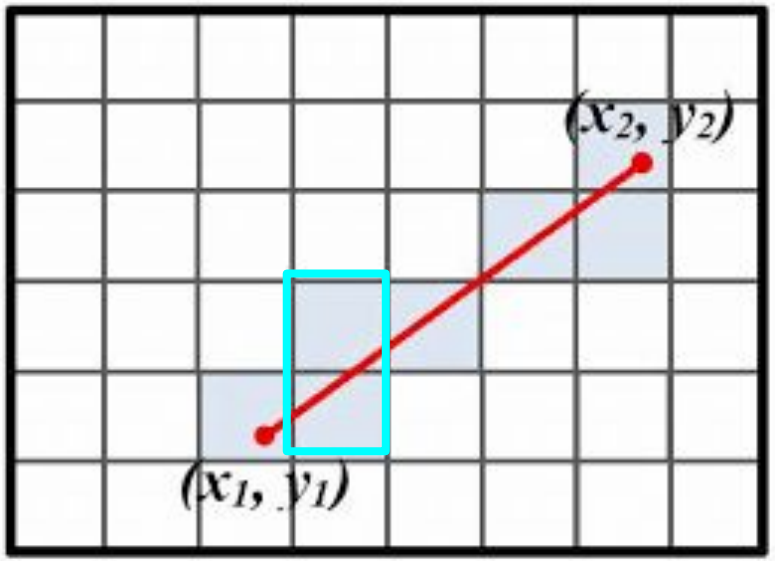
Line Drawing Algorithms (cont'd)

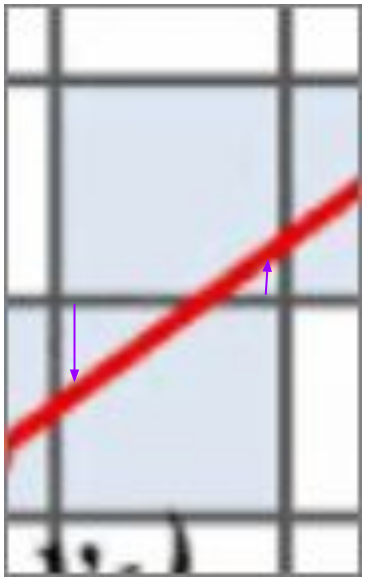
Issue:

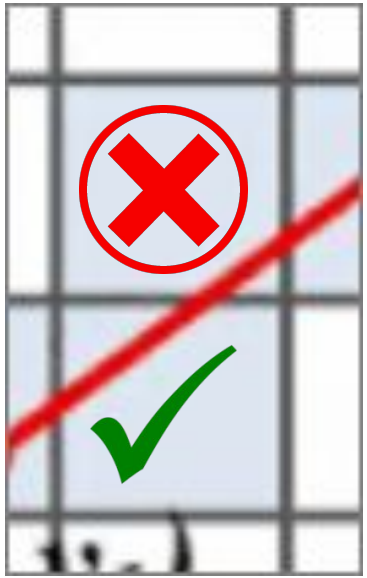
How to proportionately draw pixels?



Line Drawing Algorithms (cont'd)







Bresenham's / Xiao Lin Wu's line drawing algorithm

Rotational matrices

2D rotation

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

Uses:

- Vectors
- Minor trigonometry
- Matrix multiplication

3D rotation

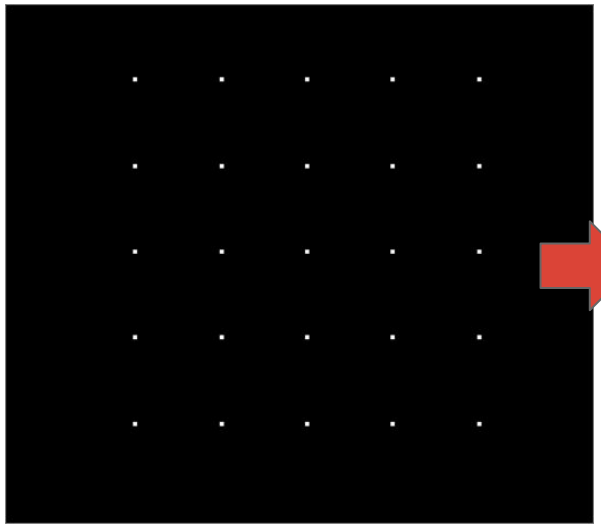
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

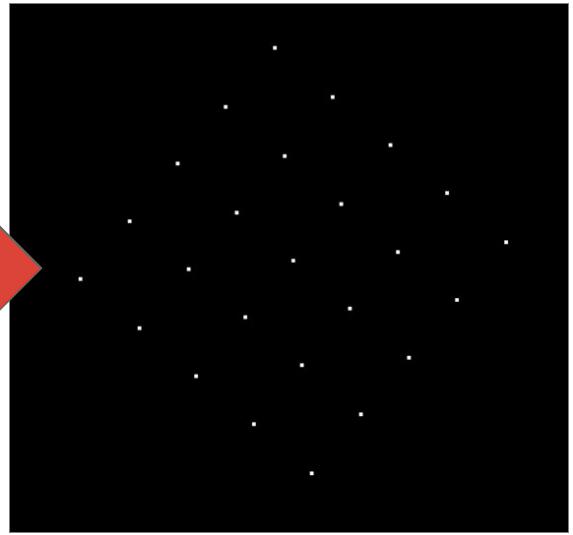
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotational matrices

2D rotation



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



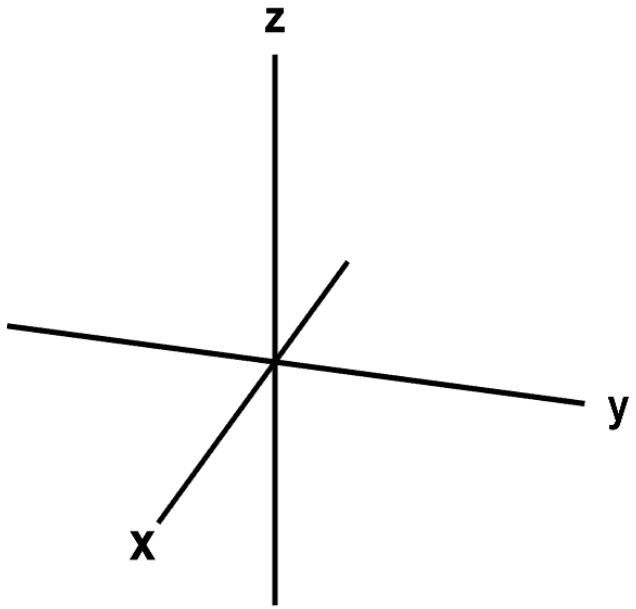
Rotational matrices (cont'd)

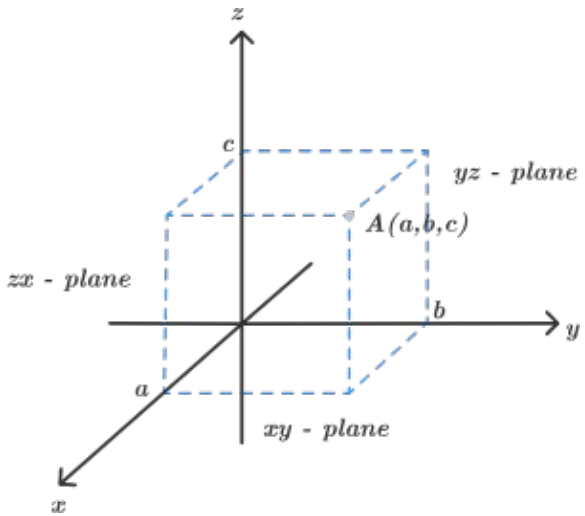
3D rotation

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

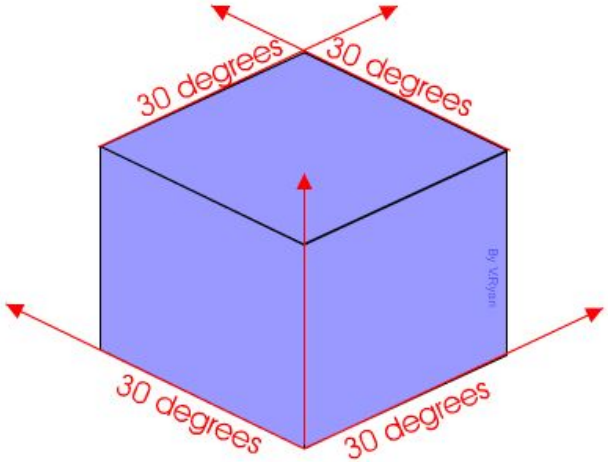




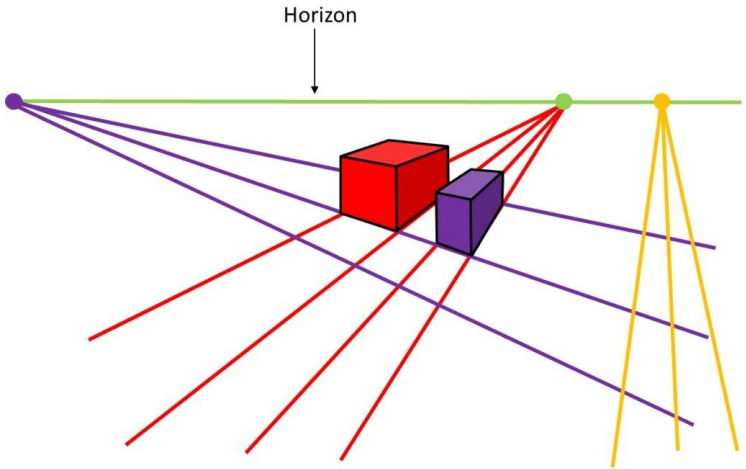
PROJECTION

Projecting 3D image in 2D screen

Isometric

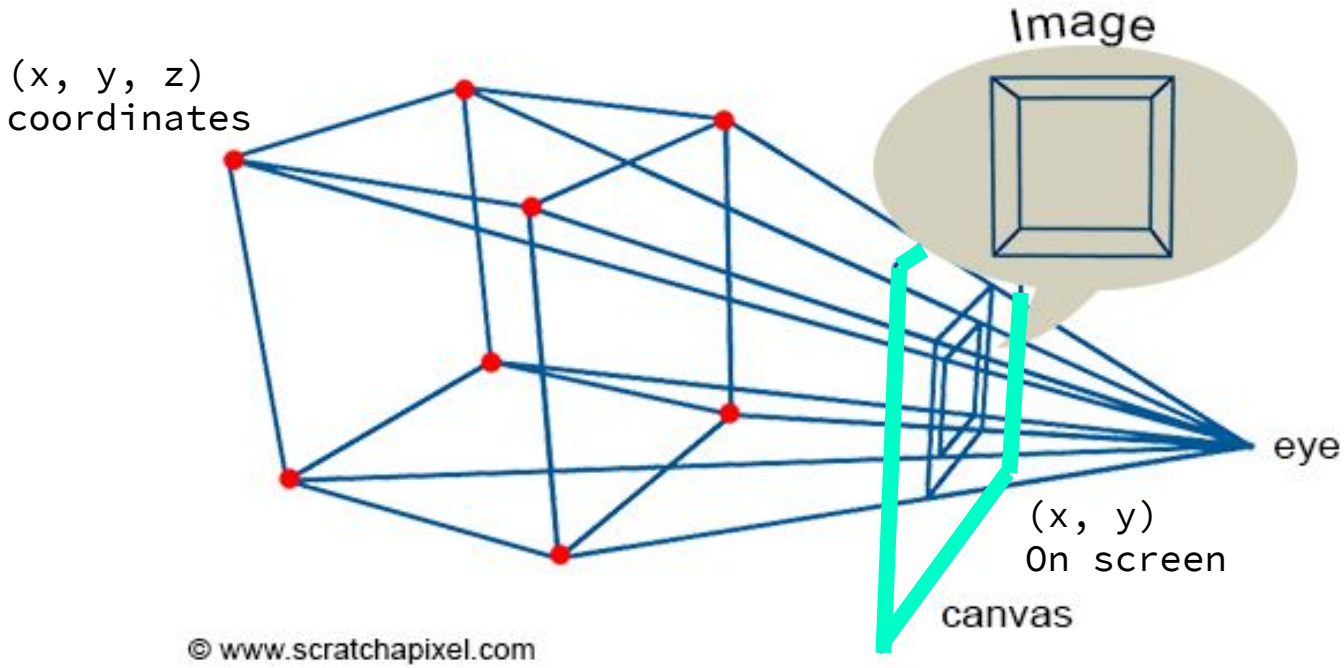


Perspective



Projection (cont'd)

Projecting 3D image in 2D screen



CONCLUSION

- Visualise objects in 3D environment
- Appreciating abstraction of 3D graphic engines
- Be comfortable with applying mathematical concepts and
- Get used to understanding something ‘just enough’ for implementation.