Introduction to Programming

Chapter 5

Input and Output (I/O)

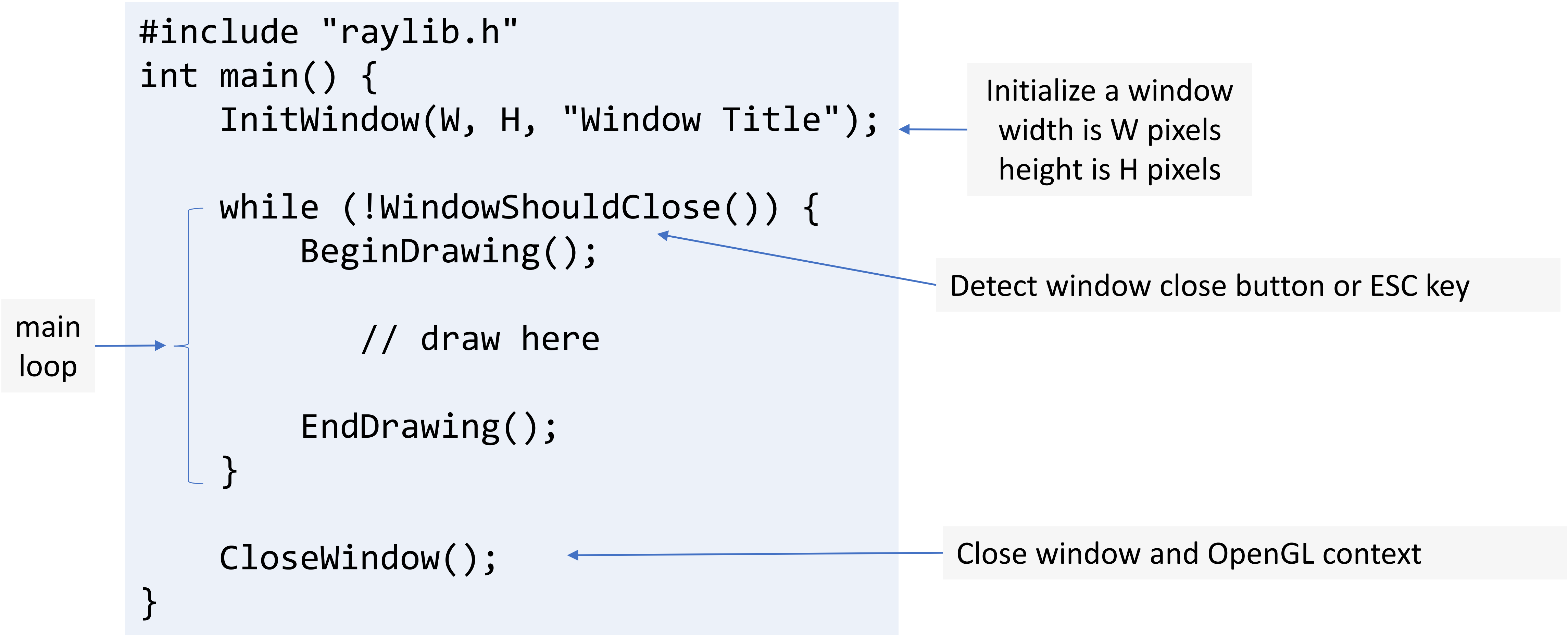
The “raylib” library

raylib
A simple and easy-to-use library
to enjoy videogames programming

https://www.raylib.com/

- Library for creating videogames in C++ (and many other languages)
- Cross platform (works on Windows, linux, macOS, android, iOS etc)
- Purely programming library
    - No GUI tools or editors (unlike Unity, Unreal Engine etc)

Structure of a program using raylib



```
#include "raylib.h"
int main() {
    InitWindow(W, H, "Window Title");

    while (!WindowShouldClose()) {
        BeginDrawing();

           // draw here

        EndDrawing();
    }

    CloseWindow();
}
```

Initialize a window
width is W pixels
height is H pixels

main
loop

Detect window close button or ESC key

Close window and OpenGL context

raylib library

```
#include "raylib.h"
```

https://www.raylib.com/cheatsheet/cheatsheet.html

```
// draw a line
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color)

// draw a color-filled circle
void DrawCircle(int centerX, int centerY, float radius, Color color)

// Draw circle outline
void DrawCircleLines(int centerX, int centerY, float radius, Color color)

// draw a color-filled rectangle
void DrawRectangle(int posX, int posY, int width, int height, Color color)

// Draw rectangle outline
void DrawRectangleLines(int posX, int posY, int width, int height, Color color)

// draw text (using default font)
void DrawText(const char *text, int posX, int posY, int fontSize, Color color)

// Load texture from file into GPU memory (VRAM)
Texture2D LoadTexture(const char *fileName);

// Draw a Texture2D
void DrawTexture(Texture2D texture, int posX, int posY, Color tint);
```

... and hundreds more

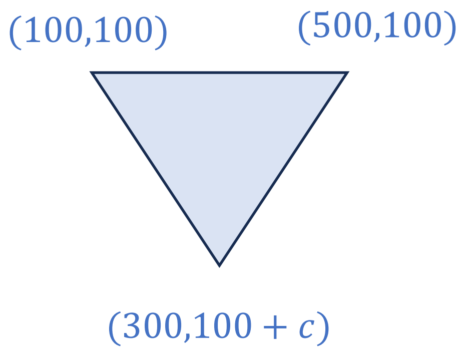
Hello raylib

```
int c = 400*sqrt(3.0) / 2.0;
InitWindow(600, 600, "Hello raylib");
while (!WindowShouldClose()) {

    BeginDrawing();
        DrawLine(100, 100, 500, 100, WHITE);
        DrawLine(100, 100, 300, 100+c, WHITE);
        DrawLine(300, 100+c, 500, 100, WHITE);

        DrawText("Hello World!", 200, 200, 35, BLUE);

        DrawCircle(300,300, 10, RED);
    EndDrawing();
}
CloseWindow();
```

c is height of equilateral triangle with base length 400

draw triangle

draw circle centered at $(300,300)$ of radius 10



Running raylib program

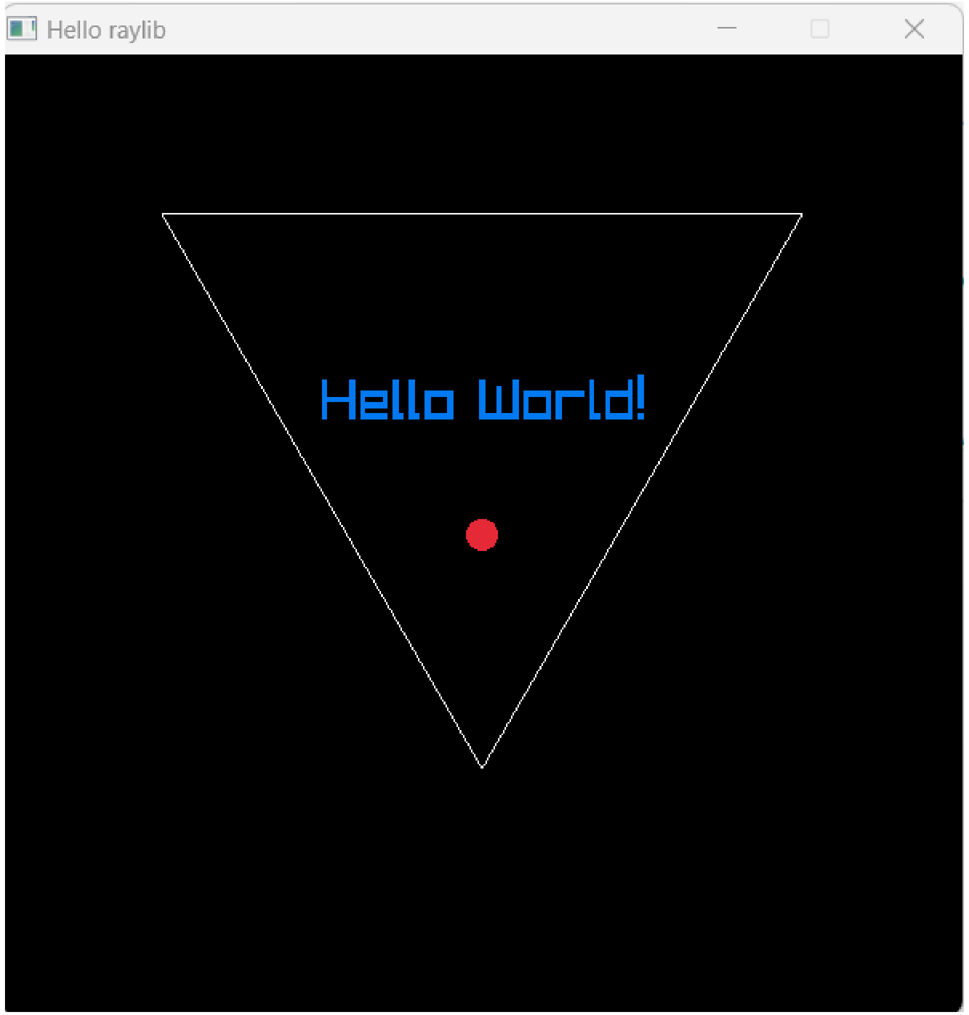
Install raylib (in MSYS2): Open MSYS2 terminal and type

```
pacman –S mingw-w64-ucrt-x86_64-raylib
```

Compile program:

```
g++ hello-raylib.cpp -lraylib
```

```
BeginDrawing();
    DrawLine(100, 100, 500, 100, WHITE);
    DrawLine(100, 100, 300, 100+c, WHITE);
    DrawLine(300, 100+c, 500, 100, WHITE);

    DrawText("Hello World!", 200, 200, 35, BLUE);

    DrawCircle(300,300, 10, RED);
EndDrawing();
```

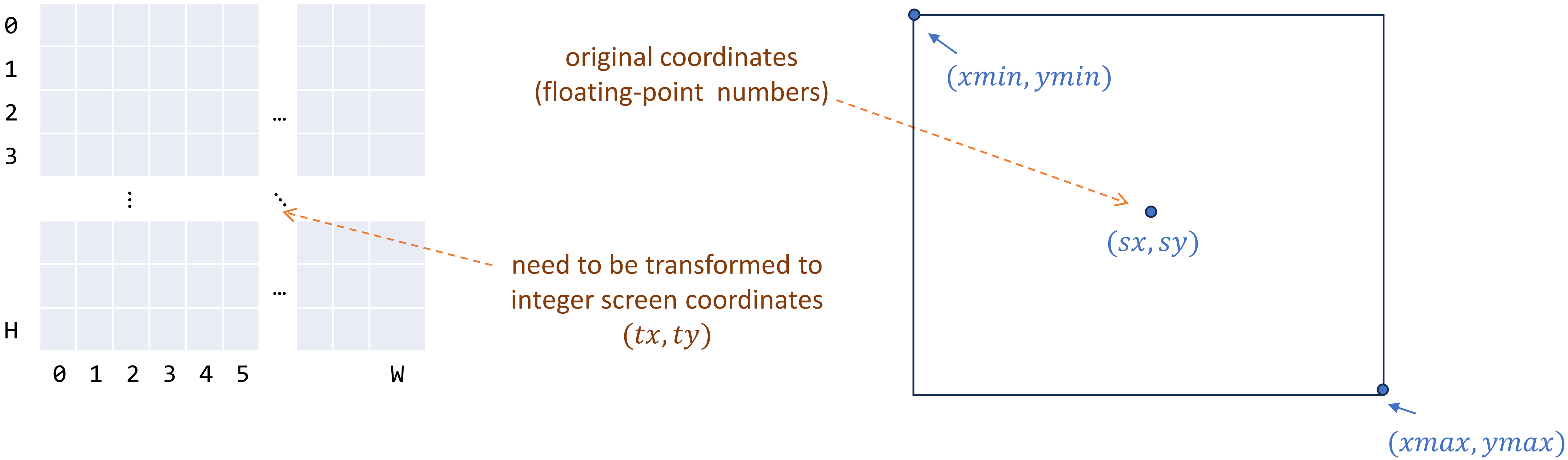


Hello World!

Coordinate transformation

Coordinates of raylib window of width W+1 and height H+1

Coordinates of something we would like to draw



original coordinates (floating-point numbers)

need to be transformed to integer screen coordinates (tx, ty)

Transformation (sx,sy) → (tx,ty)

$$tx = round\left(\frac{sx - xmin}{xmax - xmin} \times W\right)$$

$$ty = round\left(\frac{sy - ymin}{ymax - ymin} \times H\right)$$

An application: data visualization

```
// width, height, max number of points in the input
const int W=800, H=500, N=13510; int x[N], y[N];

// read coordinates of bounding box from standard input
double xmin, xmax, ymin, ymax;
std::cin >> xmin >> xmax >> ymin >> ymax;

// read points from standard input
double px, py; int count = 0;
while(std::cin >> px >> py) {
    x[count] = round((px-xmin)/(xmax-xmin) * W);
    y[count] = round((py-ymin)/(ymax-ymin) * H);
    count++;
}
```

read longitude latitude pair and transform to screen coordinates

```
InitWindow(W+1, H+1, "Plot corrdinates");
while (!WindowShouldClose()) {
    BeginDrawing();
        // fill window with WHITE
        ClearBackground(WHITE);
        for(int i=0; i<count; i++)
            DrawCircle(x[i],y[i],2,BLUE);
    EndDrawing();
}
CloseWindow();
```

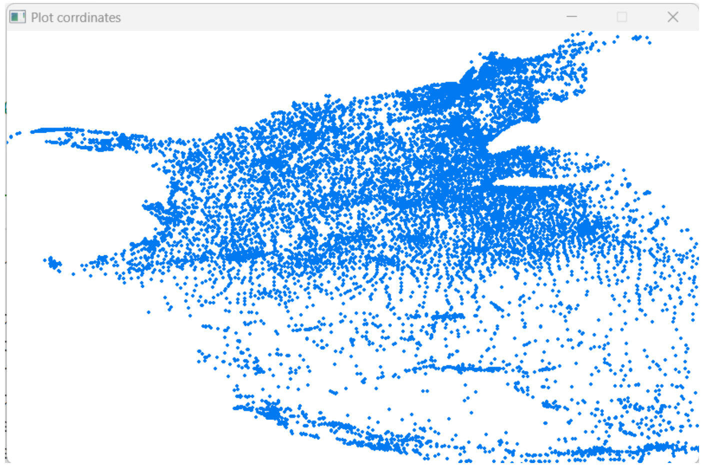
```
usa13509.txt
```

```
245552 490000
669905 1244962

245552.7780 817827.7780
247133.3330 810905.5560
247205.5560 810188.8890
249238.8890 806280.5560
250111.1110 805152.7780
...
```

min_x   max_x
min_y   max y

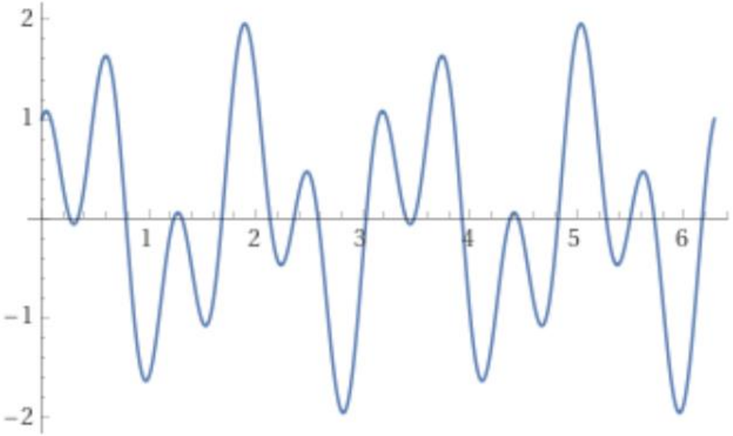
longitude latitude list of 13509 US cities

```
plot-filter.exe < usa13509.txt
```



`raylib` application: plotting a function

Goal. Plot $\mathrm{f}(\mathrm{x}) = \sin 4x + \cos 10x$ in the interval $(0,2\pi)$



Method:

- Evaluate function on $N+1$ values between 0 and 2π
- For every i, draw a line between (x_i, y_i) and (x_{i+1}, y_{i+1})

Step 1: Function evaluation and coordinates transformation

$\boldsymbol{i}$	0	1	…	i	…	N
$\boldsymbol{x_i}$	$2\pi \times \frac{0}{N}$	$2\pi \times \frac{1}{N}$	…	$2\pi \times \frac{i}{N}$	…	$2\pi \times \frac{N}{N}$
$\boldsymbol{y_i}$	$f(x_0)$	$f(x_1)$	…	$f(x_i)$	…	$f(x_N)$

```
int x[N+1], y[N+1];
for(int i=0; i<=N; i++) {
    double px = i*2*pi/N;
    double py = sin(4*px) + cos(10*px);
    x[i] = round((px-xmin)/(xmax-xmin) * W);
    y[i] = round((py-ymin)/(ymax-ymin) * H);
}
```

Step 2: Draw lines

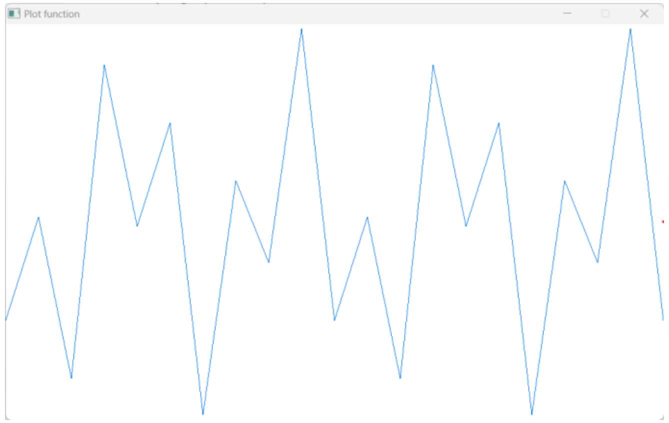
```
ClearBackground(WHITE);
for(int i=0; i<N; i++)
    DrawLine(x[i], y[i], x[i+1], y[i+1], BLUE);
```

raylib application: plotting a function

```
const int W=800, H=600;
const double pi=3.1415927;
const double xmin=0, xmax=2*pi, ymin=-2, ymax=2;
const int N = 200; // No. of sampling points

int x[N+1], y[N+1];
for(int i=0; i<=N; i++) {
    double px = i*2*pi/N;
    double py = sin(4*px) + cos(10*px);
    x[i] = round((px-xmin)/(xmax-xmin) * W);
    y[i] = round((py-ymin)/(ymax-ymin) * H);
}

InitWindow(W+1, H+1, "Plot function");
while (!WindowShouldClose()) {
    BeginDrawing();
        ClearBackground(WHITE);
        for(int i=0; i<N; i++)
            DrawLine(x[i], y[i], x[i+1], y[i+1], BLUE);
    EndDrawing();
}
CloseWindow();
```

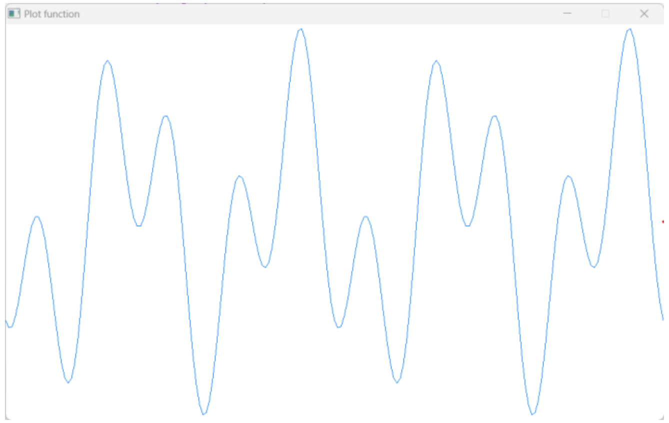
compute (xi,yi)

transform to screen coordinates



sample size N=20

Lesson 1: Plotting is easy



sample size N=200

Lesson 2: Take a sufficiently large sample—otherwise you might miss something!

Animation

To create animation with `raylib`

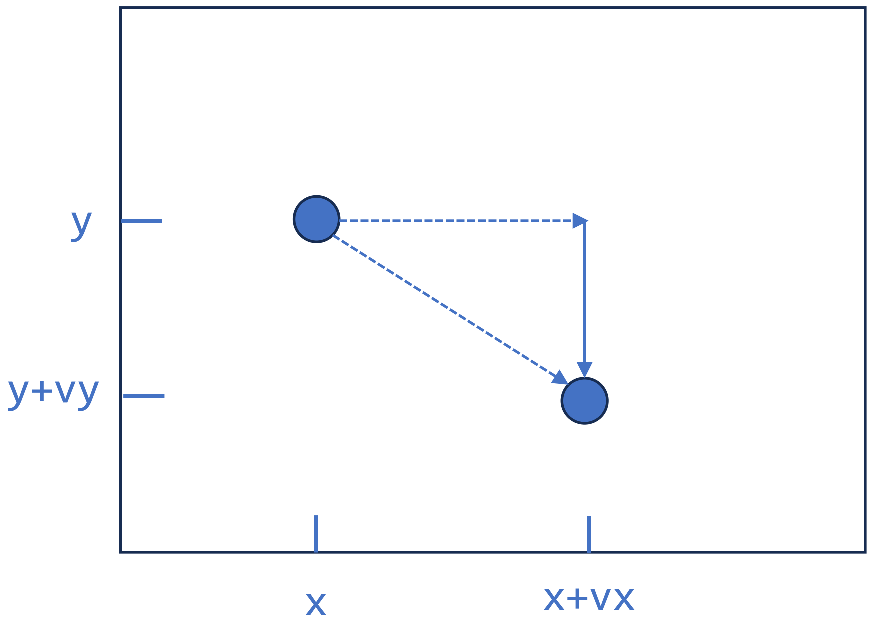
- Set target frames per second (FPS)
- Repeat the following:
    - Clear the screen.
    - Move the object.
    - Draw the object.

When display time is much greater than the screen-clear time, we have the illusion of motion.

```
int main() {
    InitWindow(W, H, "Window Title");
    SetTargetFPS(60);

    while (!WindowShouldClose()) {
        // update objects coordinates here

        BeginDrawing();
        ClearBackground(WHITE);
        // draw here

        EndDrawing();
    }

    CloseWindow();
}
```

Animation: bouncing ball

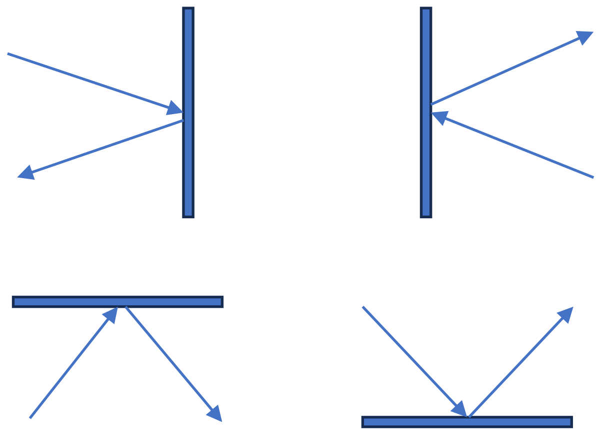
Ball has position (x, y) and
constant velocity (vx, vy).



To move the ball:
update position to $(x + vx, y + vy)$.

If the ball hits a vertical wall:
set vx to $-vx$.

If the ball hits a horizontal wall:
set vy to $-vy$.



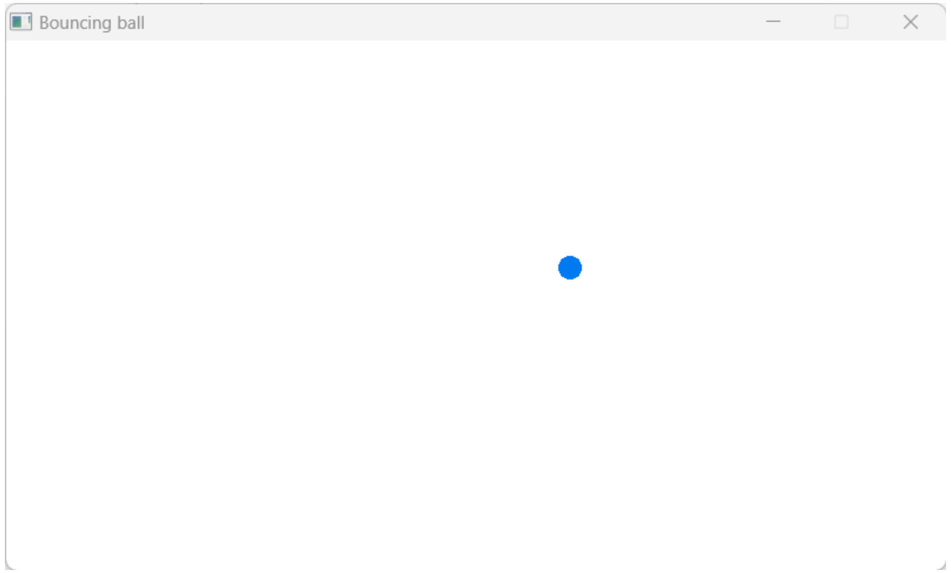
Animation: bouncing ball

```
const int W=800, H=450;
int x=W/2, y=H/2;
int vx=5, vy=4, r=10;

InitWindow(W, H, "Bouncing ball");
SetTargetFPS(60); // draw 60 frames per seconds

while (!WindowShouldClose()) {
    x = x + vx;
    y = y + vy;
    if(x+r >= W || x-r <= 0)
        vx = -vx;
    if(y+r >= H || y-r <= 0)
        vy = -vy;

    BeginDrawing();
        ClearBackground(WHITE);
        DrawCircle(x, y, r, BLUE);
    EndDrawing();
}
CloseWindow();
```

update ball position

check collision with vertical wall

check collision with horizontal wall


Bouncing ball