FDF

LINGUAGEM	C
DATA INICIO	@June 18, 2024
DATA TERMINO	@August 21, 2024
STATUS	Concluded

Objetivo

→ The result must be displayed using an isometric projection

→ The program must display a window with an image

→ The window management must be smooth

→ Pressing ESC and clicking the close button should close the window and exit the program cleanly

→ Using images from the MiniLibX library is mandatory

O que fazer?

STEP 1 → Understand the MiniLibX library

STEP 2 → Read the file

STEP 3 → Algorithm to draw the lines

STEP 4 → Draw the lines between points

STEP 5 → Add 3D (Isometric View)

Minilibx

→ Funções de Inicialização

→ mlx_init()

→ Initialize the mlx library

→ It must be called before any other function.

→ It returns NULL if initialization fails

→ mlx_new_window

→ Create a new window

→ It will return a pointer to the window

💡
```
void *mlx_ptr  //the mlx instance pointer;
int  size_x       //the width of the window;
int  size_y       //the height of the window;
char
title         // the title of the window;
return (void)
        //the window instance pointer.

void *mlx_new_window(void *mlx_ptr, int size_x, int size_y, char *title);
```

→ mlx_destroy_window

→ Destroy the window;

💡
```
void *mlx_ptr  //the mlx instance;
void *win_ptr  //the window instance;
return (int)      //has no return value (bc).

int          mlx_destroy_window(void *mlx_ptr, void *win_ptr);
```

→ Funções úteis

→ mlx_string_put

→ Place a string at location (x, y) on the provided window.

💡
```
void *mlx_ptr       //the mlx instance;
int  x              //the x location;
int  y              //the y location;
int  color          //the font color;
char *string        //the text to write;
return (int)        //has no return value (bc).

int         mlx_string_put(void *mlx_ptr, void *win_ptr, int x, int y, int
color, char *string);
```

→ Funções de imagem

→ mlx_new_image

→ Create a new image compatible with MLX.

→ This is the recommended way to buffer the image we are rendering.

💡
```
void mlx_ptr   //the mlx instance pointer;
int  width     //the width of the image to be created;
int  height    //the height of the image to be created;
return (void *)
               //the image instance reference.

void *mlx_new_image(void *mlx_ptr,int width,int height);
```

→ mlx_get_data_addr

→ Get the memory address of the provided image.

→ To get or set the pixel value (5, 100) in an image size of (500, 500), we would need to locate the position as follows:

```
int pos = (y * size_line + x * (bits_per_pixel / 8));
```

→ As we need to align the bits before writing, we need to do the following for the best result:

```
char *mlx_data_addr = mlx_get_data_addr();
*(unsigned int *)mlx_data_addr = color;
```

→mlx_put_image_to_window

→ Put an image on the provided window at position (x, y).

→ This is the recommended way to write large amounts of graphical data all at once.

→ mlx_destroy_image

→ Destroy the image.

```
void *mlx_ptr     //the mlx instance;
void *img_ptr     //the image instance;
return (int)      //has no return value (bc).

int       mlx_destroy_image(void *mlx_ptr, void *img_ptr);
```

→ Ganchos

→ mlx_loop_hook

→ Connect to the event loop.

💡 void *mlx_ptr //the mlx instance;
int (*f)() //the handler function, will be called as follows: (*f)(void *param);
void *param //the parameter to give on each event;
return (int) //has no return value (bc).

int mlx_loop_hook(void *mlx_ptr, int (*f)(), void *param);

→ mlx_loop

→ Loop over the provided MLX pointer. Each registered hook will be called in the order they were registered.

💡 void *mlx_ptr //the mlx instance;
return (int) // has no return value (bc).

int mlx_loop(void *mlx_ptr);

Algoritmo Bresenham

STEP 1

$$\Delta x = x2 - x1$$

STEP 2

$$Bigger = (mod(\Delta x), mod(\Delta y))$$

STEP 3

$$Pa = \Delta x / Bigger$$

$$Pb = \Delta y / Bigger$$

STEP 4

- x1 → x2 ⟶ Pa
- y1 → y2 ⟶ Pb

Let's continuously add Pa and Pb to the initial point until we reach the final point.

Vista Isométrica

- **Rotation of 35.264° (arctan(sqrt(2)/2)) around the xx axis**:
  - This angle is calculated as arctan(2) ≈ 35.264°.
- **Rotation of 45° around the yy axis**:
  - This angle is 45°.
- **Rotation of 0° around the zz axis**:
  - There is no rotation around the z axis in classical isometric projection.

```
void    isometric(t_point *dot, double angle)
{
    dot->x = (dot->x - dot->y) * cos(angle);
    dot->y = (dot->x + dot->y) * sin(angle) - dot->z;
}
```

→ Projeção de ponto

Calculate for each point:

$$x(2D) = x' - z'$$

$$y(2D) = y' - z'$$

Draw map

```
void    draw(t_point **matri
{
    int     y;
    int     x;
```

To print the map centered in the middle of the window, we need to subtract half of the map's width from x and half of the map's height from y

```
    y = 0;
    while (matrix[y])
    {
        x = 0;
        while (1)
        {
            if (matrix[y + 1
                draw_line(ma
            if (!matrix[y][x
                draw_line(ma
            if (matrix[y][x]
                break ;
            x++;
        }
        y++;
    }
    push_image_to_window(dat
    menu(data);
}
```

Translation

→ Add two variables to the structure: direction y and direction x

> a→x += WIN_WIDTH / 2 + x_translate;

> b→x += WIN_WIDTH / 2 + x_translate;

> a→y += WIN_HEIGHT / 2 + y_translate;

> b→y += WIN_HEIGHT / 2 + y_translate;

Zoom

The same applies to zoom. Choose a pair of keys that, whenever pressed, will increase or decrease the zoom factor mentioned earlier, depending on whether you want to zoom in or out.