FdF Cookbook

Watch

FdF Graphics MiniLibX

Altruist vbrazhni

Altruist vbrazhni 2295 2018-08-23

Foreword

You can also read this information on my GitHub **wiki-pages**. Maybe it will be more comfortable for someone.

Link: https://github.com/VBrazhnik/FdF/wiki 494

How to handle mouse buttons and key presses?

On macOS, if you want to handle mouse buttons and key presses and close the window with the red button, you can use the following function instead of the other hook functions:

```
int mlx_hook(void *win_ptr, int x_event, int x_mask, int (*funct)(), void *param);
```

To handle a key press

At the place of `int x_event` parametr use 2.

At the place of `int (*funct)()` parameter you use the following function:

```
int key_press(int keycode, void *param)
```

To handle a key release

At the place of `int x_event` parametr use 3.

At the place of `int (*funct)()` parameter you use the following function:

```
int key_release(int keycode, void *param)
```

To handle a mouse button press

At the place of `int x_event` parametr use 4.

At the place of `int (*funct)()` parameter you use the following function:

```
int mouse_press(int button, int x, int y, void *param)
```

To handle a mouse button release

At the place of `int x_event` parametr use 5.

At the place of `int (*funct)()` parameter you use the following function:

```
int mouse_release(int button, int x, int y, void *param)
```

To handle a mouse movement

At the place of `int x_event` parametr use `6`.

At the place of `int (*funct)()` parameter you use the following function:

```
int mouse_move(int x, int y, void *param)
```

To handle an expose event

At the place of `int x_event` parametr use `12`.

At the place of `int (*funct)()` parameter you use the following function:

```
int expose(void *param)
```

To handle a red button (`X` button) press

At the place of `int x_event` parametr use `17`.

At the place of `int (*funct)()` parameter you use the following function:

```
int close(void *param)
```

Complete `int close(void *param)` function:

```
int close(void *param)
{
    (void)param;
    exit(0);
}
```


Tip:

`x_mask` is ignored on macOS. But if you want that your`FdF` will have compatibility with Linux, you must use`x_mask`.

Key codes

Mouse button codes

- Left button — `1`
- Right button — `2`
- Third (Middle) button — `3`

- Scroll Up — 4
- Scroll Down — 5
- Scroll Left — 6
- Scroll Right — 7

Masks

You can find values of `x_mask` here 544.

Tip:

`x_mask` for `int close(void *param)` is `(1L << 17)`.

Created by vbrazhni **2018-08-23 12:29** — 1.7K words , with a read time of **5 minutes**
Posted in cursus **42** — Written in **English**

Users in the topic

Links in the topic

Data Definitions for libX11547 refspecs.linuxfoundation.org
Pages · VBrazhnik/FdF Wiki · GitHub495 github.com
Bresenham's line algorithm - Wikipedia309 en.wikipedia.org
Rotation matrix - Wikipedia295 en.wikipedia.org
Xiaolin Wu's line algorithm - Wikipedia255 en.wikipedia.org
Вращение фигуры в 3-х мерном пространстве | Компьютерная графика231 grafika.me
Isometric 2:1 Projections: Isometric Infographic Vectors - Vectips189 vectips.com
kirupa.com - Isometric Transformation177 kirupa.com
How do I calculate color gradients? - Graphic Design Stack Exchange139 graphicdesign.stackexchange.com
Geometry (How Does Matrix Work: Part 1)138 scratchapixel.com

Altruist vbrazhni

Altruist vbrazhni 2165 2018-08-23

How to draw a line?

To draw a line you can use Bresenham's line algorithm 281 (simpler solution) or Xiaolin Wu's line algorithm 236 (more sophisticated solution which will produced more beautiful result).

Reply to this message...

Altruist vbrazhni

Altruist vbrazhni 2018-08-23

How to create linear gradient?

Here we will consider how to find the color between any two color points through linear interpolation.

First of all we need to find current point position between two points with known colors. Position value must be expressed in percentages.

The following function will help you find this value:

```
double percent(int start, int end, int current)
{
    double placement;
    double distance;

    placement = current - start;
    distance = end - start;
    return ((distance == 0) ? 1.0 : (placement / distance));
}
```

You can calculate this value depending on which delta value is bigger. Delta between `x` values of known points or delta between `y` values.

Part of code:

```
// ...
    double percentage;

    if (delta.x > delta.y)
        percentage = percent(start.x, end.x, current.x);
    else
        percentage = percent(start.y, end.y, current.y);
// ...
```

Then for creating each light (**R**ed, **G**reen, **B**lue) we need to get light from start and end point and use linear interpolation. At the end we need to get new color by union red, green and blue light.

Part of code:

```
// ...
    int red;
    int green;
    int blue;

    // Get percentage

    red = get_light((start.color >> 16) & 0xFF, (end.color >> 16) & 0xFF, percentage);
    green = get_light((start.color >> 8) & 0xFF, (end.color >> 8) & 0xFF, percentage);
    blue = get_light(start.color & 0xFF, end.color & 0xFF, percentage);
    return ((red << 16) | (green << 8) | blue);
```

```
int get_light(int start, int end, double percentage)
{
    return ((int)((1 - percentage) * start + percentage * end));
}
```

Complete code:

```
int get_light(int start, int end, double percentage)
{
    return ((int)((1 - percentage) * start + percentage * end));
}

int get_color(t_point current, t_point start, t_point end, t_point delta)
{
    int     red;
    int     green;
    int     blue;
    double  percentage;

    if (current.color == end.color)
```

```
        return (current.color);
    if (delta.x > delta.y)
        percentage = percent(start.x, end.x, current.x);
    else
        percentage = percent(start.y, end.y, current.y);
    red = get_light((start.color >> 16) & 0xFF, (end.color >> 16) & 0xFF, percentage);
    green = get_light((start.color >> 8) & 0xFF, (end.color >> 8) & 0xFF, percentage);
    blue = get_light(start.color & 0xFF, end.color & 0xFF, percentage);
    return ((red << 16) | (green << 8) | blue);
}
```

Basic information was found here 126.

Color for pixel

Everything is easy if you decided to use the following function:

```
int mlx_pixel_put(void *mlx_ptr, void *win_ptr, int x, int y, int color);
```

In this case, the order of lights is standard:

0	R	G	B
8 bits	8 bits	8 bits	8 bits

As you can see that the first byte is filled with zeros. It means that the alpha channel of color is not supported by `minilibx`.

You can find this information in `mlx_pixel_put` man file.

Also, this information is actual for color parameter in the function which displays text:

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

But if you decided to use an image, you will face with more complicated usage rules.

You will work with the following three functions:

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

```
char *mlx_get_data_addr(void *img_ptr, int *bits_per_pixel, int *size_line, int *endian);
```

```
int mlx_put_image_to_window(void *mlx_ptr, void *win_ptr, void *img_ptr, int x, int y);
```

And the most interesting is the second function with such parameters as `bits_per_pixel` and `endian`.

> **What is bits per pixel or bit-depth value?**
>
> The number of bits used to define a pixel's color shade is its bit-depth. True color is sometimes known as 24-bit color. Some new color display systems offer a 32-bit color mode. The extra byte, called the alpha channel, is used for control and special effects information.

For `macOS` value of `bits_per_pixel` is constant. You can find the following lines in source files of minilibx:

```
#define UNIQ_BPP 4
```

```
// assume here 32bpp little endian

char *mlx_get_data_addr(mlx_img_list_t *img_ptr, int *bits_per_pixel, int *size_line, int *endian)
{
    *bits_per_pixel = UNIQ_BPP * 8;
    *size_line = img_ptr->width * UNIQ_BPP;
    // ...
}
```

If you decided to support only `macOS`, you don't need to worry about the size of the variable with color. It is exactly as needed — 4 bytes (32 bits).

`endian` is the most important parameter that we have to consider.

For `macOS` its value is `0`, which means `little endian`.

Information about `endian` value you can also find in source files of minilibx:

```
/*
** endian : 0 = sever X is little endian, 1 = big endian
** endian : useless on macos, client and graphical framework have the same endian
```

```
*/
```

```
// assume here 32bpp little endian

char *mlx_get_data_addr(mlx_img_list_t *img_ptr, int *bits_per_pixel, int *size_line, int *endian)
{
    // ...
    *endian = 0; // little endian for now on mac-intel
    // ...
}
```

Big-endian and little-endian are the formats of ordering bytes.

Big-endian is the format that we used to know as**normal**.

Little-endian order is **reversed**.

For color these two formats look like:

Byte number	**0**	**1**	**2**	**3**
Big endian	0	R	G	B
Little endian	B	G	R	0

So in the case of little-endian format, you have to use reversed order of color components:

```
// ...
    int i;

    i = (x * fdf->bits_per_pixel / 8) + (y * fdf->size_line);
    fdf->data_addr[i] = color; // B - Blue
    fdf->data_addr[++i] = color >> 8; // G - Green
    fdf->data_addr[++i] = color >> 16; // R - Red
    fdf->data_addr[++i] = 0; // Alpha channel
// ...
```

Reply to this message...

Altruist vbrazhni

Altruist vbrazhni 2047 2018-08-23

How to rotate figure in 3D?

If you want to rotate a vector you should construct what is known as a[rotation matrix 295](#).

X-Axis Rotation

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

After the transformations, we will get the formulas:

$$x' = x;$$
$$y' = y * cos(\theta) + z * sin(\theta);$$
$$z' = -y * sin(\theta) + z * cos(\theta);$$

Y-Axis Rotation

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

After the transformations, we will get the formulas:

$$x' = x * cos(\theta) + z * sin(\theta);$$
$$y' = y;$$
$$z' = -x * sin(\theta) + z * cos(\theta);$$

Z-Axis Rotation

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

After the transformations, we will get the formulas:

$$x' = x * cos(\theta) - y * sin(\theta);$$
$$y' = x * sin(\theta) + y * cos(\theta);$$
$$z' = z;$$

Source of information 227 (Russian)

Reply to this message...

Altruist vbrazhni

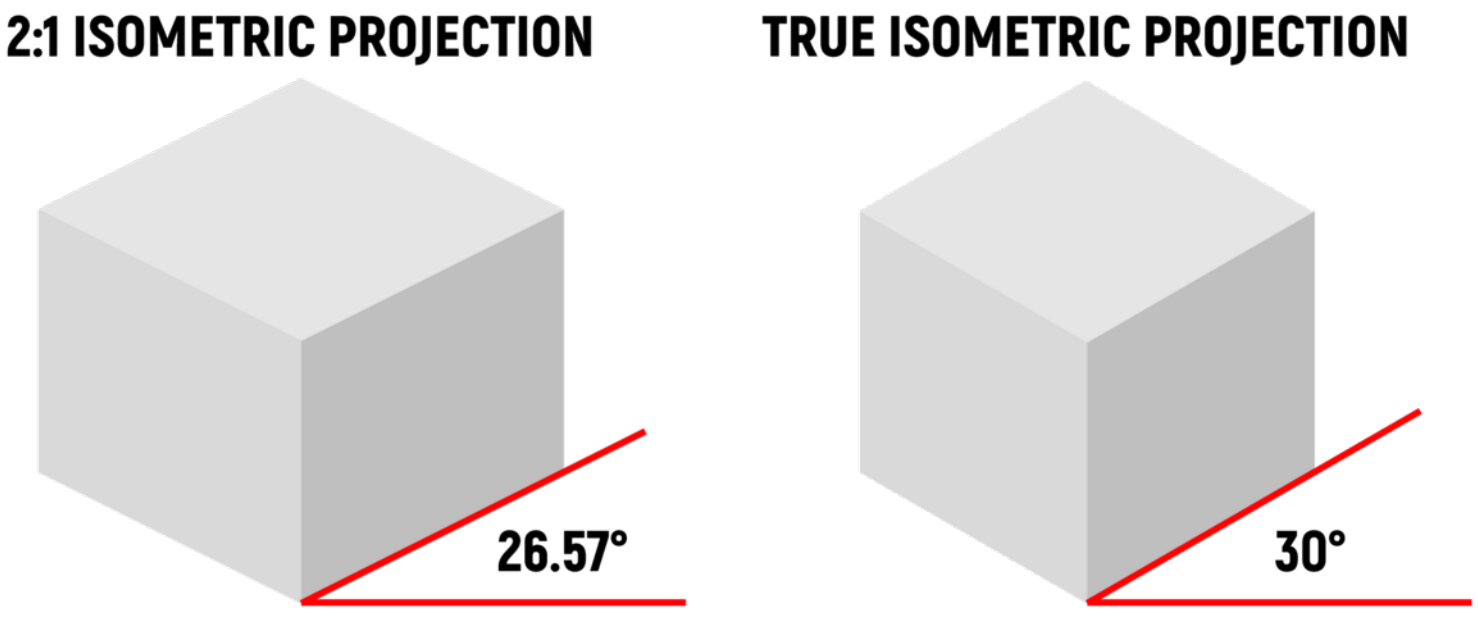
Altruist vbrazhni 2036 2018-08-23

How to perform isometric transformations?

There are **"true"** isometric projection and **2:1** isometric projection.

True isometric projection uses a 30° angle (0.523599 rad).

2:1 isometric projection uses a 26.57° angle (0.46373398 rad).



Source of information 179

Code for transforming:

```
static void iso(int *x, int *y, int z)
{
    int previous_x;
    int previous_y;

    previous_x = *x;
    previous_y = *y;
    *x = (previous_x - previous_y) * cos(0.523599);
    *y = -z + (previous_x + previous_y) * sin(0.523599);
}

t_point project(t_point p, t_fdf *fdf)
{
// ...
    if (fdf->camera->projection == ISO)
        iso(&p.x, &p.y, p.z);
// ...
}
```

Source of information 167

This transformation is pretty good, but it results in a "backwards" image. (This is most noticeable in the example that says "42", but the text is reversed.)

To fix this, invert the + and -:

```
new_x = (x + y) * cos(angle);
new_y = (x - y) * sin(angle) - z;
```

reply

2 0

2

Altruist vbrazhni

1714 2019-01-10

It depends only on your realization of reading and storage coordinates of a map. Someone will get a completely correct image after projection, someone will get a reversed image.

You are completely right that it is simply to fix. You can correct this formula and formulas in the previous publication (changing sign before x, y or z) to reverse axis direction. And this change will reverse your image by x, y or z-axis.

The format of formulas, that are listed here, is the best for my FdF. But if you get reversed images, change them to get a perfect result.

reply

7 0

7

mtaylor

1698 2019-01-10

Slanting right:

```
new_x = (x + y) * cos(angle);
new_y = (x - y) * sin(angle) - z;
```

Slanting left:

```
new_x = (x - y) * -cos(angle);
new_y = ((x + y) * sin(angle)) - z;
```

Mirror image slanting right:

```
new_x = (x - y) * cos(angle);
new_y = ((x + y) * sin(angle)) - z;
```

Mirror image slanting left:

```
new_x = (x + y) * -cos(angle);
new_y = ((x - y) * sin(angle)) - z;
```

reply

Reply to this message...

cfargere

cfargere 1316 2019-03-19

Interestingly, the rotation matrix shown here are reversed from the scratchpixel tutorialhere 108.

Reply to this message...

cyuriko

cyuriko 648 2 months ago

Most helpful, thanks!

Reply to this message...

bshara

bshara 541 2 months ago

thanks😄

Reply to this message...

jmacgyve

jmacgyve 344 a month ago

th

Reply to this message...

samymone

samymone 306 a month ago

very helpful, thanks!)

Reply to this message...

Load next messages