Rasterization extra

Ellipse rasterization

Follow the derivation of circle rasterization, I think the incremental derivation for ellipses should be:

```
D < 0
    D' = D + (2 * x + 3) * b * b
    y' = y

D >= 0
    D' = (2 * x + 3) * b * b + (2 - 2 * y) * a * a
    y' = y - 1
```

where `a` and `b` are semi-major and semi-minor axes.

Similarly, the initial D is set to be `b * b + a * a / 4 - sqrt(a * b))`. Also, we need to change `circlePoints` to be `ellipsePoints` which only duplicates four of the calculated results since it's an ellipse.

```
if (a >= b) {
    int x = 0, y = b, D = b * b + a * a / 4 - int(std::sqrt(a*b));
    ellipsePoints(x, y, x0, y0, color);
    while (y > 0) {
        if (D < 0) {
            D += (2 * x + 3)*b*b;
        }
        else {
            D += (2 * x + 3)*b*b + (2 - 2 * y)*a*a;
            y -= 1;
        }
        x += 1;
        ellipsePoints(x, y, x0, y0, color);
    }
}
else {
    std::swap(a, b);
    int x = 0, y = b, D = b * b + a * a / 4 - int(std::sqrt(a*b));
    ellipsePoints(y, x, x0, y0, color);
    while (y > 0) {
        if (D < 0) {
            D += (2 * x + 3)*b*b;
        }
        else {
            D += (2 * x + 3)*b*b + (2 - 2 * y)*a*a;
            y -= 1;
        }
        x += 1;
        ellipsePoints(y, x, x0, y0, color);
```

```
    }
}
```

Implement clipping

I'm not quite sure what this part means. The `putPixel` has already dealt with this problem quite well.

```
void putPixel(int x, int y, color_f color)
{
        // clamp
        if (x >= g_image_width || x < 0 || y >= g_image_height || y < 0) return;

        // write
        g_image[y* g_image_width + x] = color;
}
```

Support different colors

It's clear that the original code has already implemented most of the code we need to support colors.

1. We need to change `g_image's` type to represent rgb values.

```
struct color_f
{
        float r, g, b;
};
std::vector<color_f> g_image;
```

2. Change `writeImage` to pass all rgb values to `tmpData`.

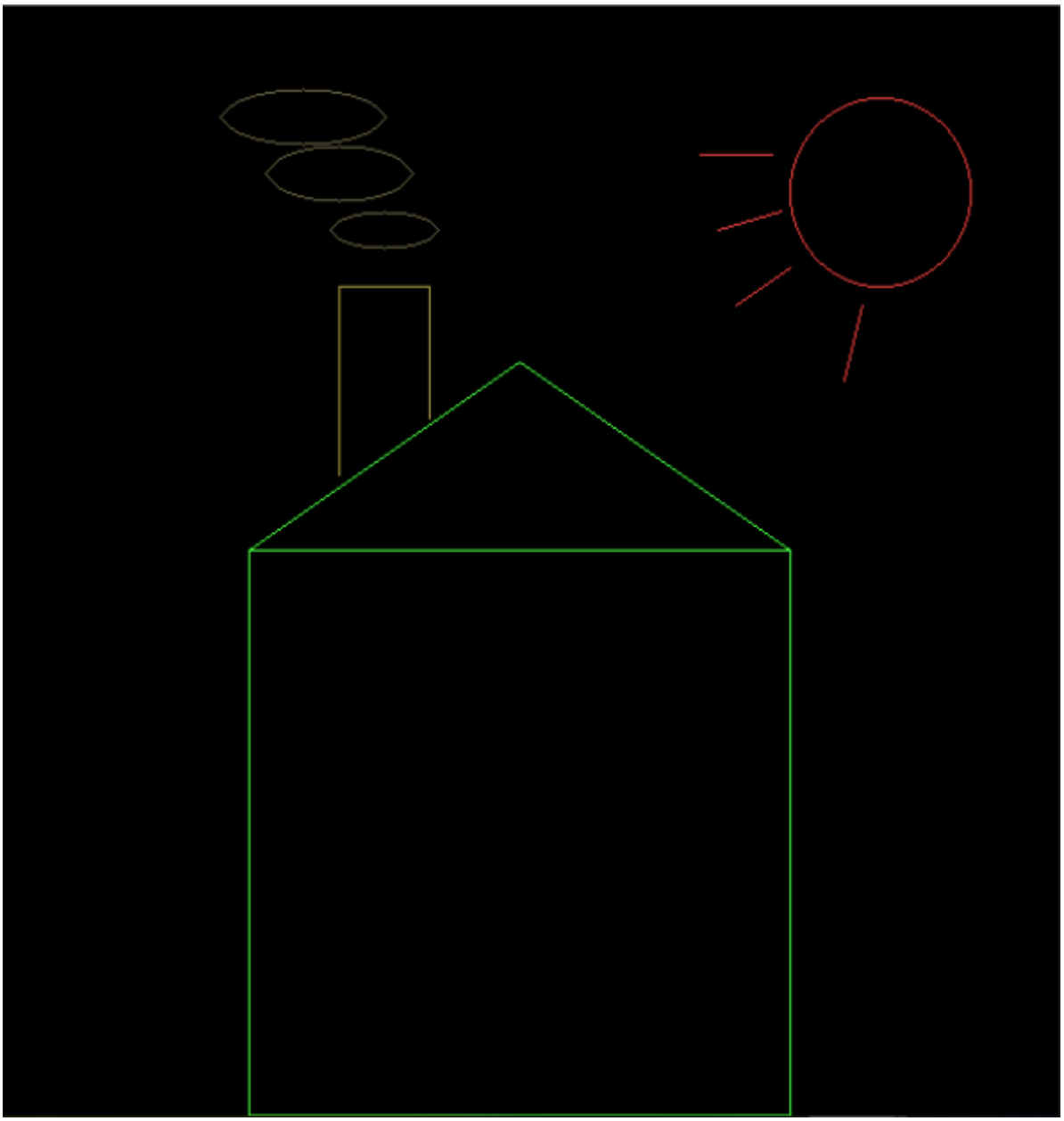
```
float tmp_r = g_image[i* g_image_width + j].r;
                        if (tmp_r < 0.0f)       tmp_r = 0.0f;
                        if (tmp_r > 1.0f)       tmp_r = 1.0f;
                        float tmp_g = g_image[i* g_image_width + j].g;
                        if (tmp_g < 0.0f)       tmp_g = 0.0f;
                        if (tmp_g > 1.0f)       tmp_g = 1.0f;
                        float tmp_b = g_image[i* g_image_width + j].b;
                        if (tmp_b < 0.0f)       tmp_b = 0.0f;
                        if (tmp_b > 1.0f)       tmp_b = 1.0f;

                        tmpData[(g_image_height - i - 1)* g_image_width + j].r =
unsigned char(tmp_r*255.0);
                        tmpData[(g_image_height - i - 1)* g_image_width + j].g =
unsigned char(tmp_g*255.0);
                        tmpData[(g_image_height - i - 1)* g_image_width + j].b =
unsigned char(tmp_b*255.0);
```

3. Change `render` to properly show the colored image.

```
void render()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glDrawPixels(g_image_width, g_image_height, GL_RGB, GL_FLOAT,
&g_image[0]);
}
```

Result



Reference

`glDrawPixels`: glDrawPixels function - Microsoft Docs