第2章光栅图形学算法上机作业

1. 实验题目

1.用DDA算法在屏幕上画一条直线。

2.采用中点线算法在屏幕上画一条直线。

3.采用中点圆算法在屏幕上画一个圆。

1. 算法描述

1.DDA 算法(Digital Differential Analyzer)

考虑将直线进行微分，由于直线各处微分相同，所以求出一个相同的 dx 和 dy，然后从起点 (sx,sy) 开始，每次位移 (dx, dy) 直到中点 (tx, ty) 为止。判断路径中的每一 个点所在像素，光栅化生成片元。

2.中点线算法(Bresenham Algorithm)

该算法是 DDA 算法的改进版本，用整数计算替换了浮点数计算，提升了性能。首先 通过对称变换将直线变成 x, y 轴都单调递增且斜率小于 45 度的直线。将 dx 取作 1，dy = (ty-sy)/(tx-sx), 将 dy 乘上 2(tx-sx) 变为整数，由于 dy/dx 小于 1，所以每次 x 增加 1 时，y 最多增加 1，所以维护 y 真分数的分子，每次加上 2(ty-sy)，一旦变为假分数，则 y 增加1， y 的真分数分子减去分母 2(tx-sx)。

3. 中点圆算法

中点圆算法是 Bresenham Algorithm 在圆上的应用，先将圆对称到第一象限 45 度以上的部分，dx=1, dy=(2x-1)/(2y-1), 然后整数化之后，设定初始分子为 (4r+1), 每次x增加 1分子减去 (4x-2), 一旦分子小于 0 则y 减去1，然后分子加上 (4y-2)。

1. 绘图代码部分

由于光栅化部分已经由 OpenGL 内置实现了，不可修改，这里直接调用 OpenGL 传入顶点集合来模拟实现。

本次实验所有代码均基于 rust 语言及其经过安全性包装的 openGL 库 glium。以及我自行编写的 rust 库 gl，用于方便 glium 的调用，其代码可以在附件中文件夹 gl 中找到。

以下是这个实验的所有源代码，也可以查看附件中 Chapter2/src/main.rs。

#[macro\_use]

extern crate glium;

use gl::camera;

use gl::shader;

use gl::action;

use gl::vertex;

use gl::vertex::Vertex;

use glium::Display;

use glium::vertex::VertexBufferAny;

fn abs(x: i32) -> i32 { if x < 0 {-x} else {x} }

// DDA 算法

fn algorithm\_dda(display: &Display, begin: [i32; 2], end:[i32; 2]) -> VertexBufferAny {

let mut ans = Vec::new();

let step = std::cmp::max( abs(begin[0] - end[0]), abs(begin[1] - end[1]) );

let dx = (end[0] - begin[0]) as f32 / step as f32;

let dy = (end[1] - begin[1]) as f32 / step as f32;

let [mut x, mut y] = [begin[0] as f32, begin[1] as f32];

let (w, h) = display.get\_framebuffer\_dimensions();

for \_ in 0..step{

ans.push( Vertex::new\_2d(x/ w as f32, y / h as f32) );

[x, y] = [x + dx, y + dy];

}

vertex::from\_vertex(&display, &ans)

}

// 中点线算法

fn algorithm\_bresenham(display: &Display, mut begin: [i32; 2], mut end:[i32; 2]) -> VertexBufferAny {

let mut ans = Vec::new();

let steep = abs(begin[0] - end[0]) < abs(begin[1] - end[1]);

if steep {

begin = [begin[1], begin[0]];

end = [end[1], end[0]];

}

if begin[0] > begin[1] {

(begin, end) = (end, begin);

}

let dx = end[0] - begin[0];

let dy = abs(end[1] - begin[1]);

let mut diff = dx / 2;

let stepy = if end[1] < begin[1] {-1} else {1};

let (mut x,mut y) = (begin[0], begin[1]);

let (w, h) = display.get\_framebuffer\_dimensions();

for \_ in begin[0]..end[0] {

diff -= dy;

if diff < 0{

y += stepy;

diff += dx;

}

let (px, py) = if steep {(y, x)} else {(x, y)};

ans.push( Vertex::new\_2d(px as f32/ w as f32, py as f32 / h as f32) );

x += 1;

}

vertex::from\_vertex(&display, &ans)

}

// 中点圆算法

fn algorithm\_bresenham\_circle(display: &Display, c: [i32; 2], r: i32 ) -> VertexBufferAny {

let mut ans = Vec::new();

let [mut x, mut y] = [0, r];

let mut diff = 4 \* r + 1;

let (w, h) = display.get\_framebuffer\_dimensions();

while x < y {

ans.push( Vertex::new\_2d((c[0] + x) as f32 / w as f32, (c[1] + y) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] - x) as f32 / w as f32, (c[1] + y) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] + x) as f32 / w as f32, (c[1] - y) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] - x) as f32 / w as f32, (c[1] - y) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] + y) as f32 / w as f32, (c[1] + x) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] - y) as f32 / w as f32, (c[1] + x) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] + y) as f32 / w as f32, (c[1] - x) as f32 / h as f32) );

ans.push( Vertex::new\_2d((c[0] - y) as f32 / w as f32, (c[1] - x) as f32 / h as f32) );

x += 1;

diff -= 4 \* x - 2;

if diff < 0 {

y -= 1;

diff += 4 \* y - 2;

}

}

vertex::from\_vertex(&display, &ans)

}

fn main() {

#[allow(unused\_imports)]

use glium::{glutin, Surface};

let event\_loop = glutin::event\_loop::EventLoop::new();

let wb = glutin::window::WindowBuilder::new();

let cb = glutin::ContextBuilder::new().with\_depth\_buffer(24);

let display = glium::Display::new(wb, cb, &event\_loop).unwrap();

let program = shader::get\_default\_shader(&display);

let vertex\_buffer1 = algorithm\_dda(&display, [0, 0], [300, 500]);

let vertex\_buffer2 = algorithm\_bresenham(&display, [0, -400], [300, 300]);

let vertex\_buffer3 = algorithm\_bresenham\_circle(&display, [0, 0], 233);

let indices\_buffer = glium::index::NoIndices(glium::index::PrimitiveType::Points);

action::start\_loop(event\_loop, move |events| {

let mut target = display.draw();

target.clear\_color\_and\_depth((0.0, 0.0, 0.0, 1.0), 1.0);

let uniforms = uniform! {

perspective: camera::CameraState::flat\_perspective(),

view: camera::CameraState::flat\_view(),

model: [

[1.0, 0.0, 0.0, 0.0],

[0.0, 1.0, 0.0, 0.0],

[0.0, 0.0, 1.0, 0.0],

[0.0, 0.0, 0.0, 1.0f32],

]

};

let params = glium::DrawParameters {

depth: glium::Depth {

test: glium::draw\_parameters::DepthTest::IfLess,

write: true,

.. Default::default()

},

.. Default::default()

};

target.draw(&vertex\_buffer1, &indices\_buffer, &program, &uniforms, &params).unwrap();

target.draw(&vertex\_buffer2, &indices\_buffer, &program, &uniforms, &params).unwrap();

target.draw(&vertex\_buffer3, &indices\_buffer, &program, &uniforms, &params).unwrap();

target.finish().unwrap();

let mut action = action::Action::Continue;

// handling the events received by the window since the last frame

for e in events {

match e {

glutin::event::Event::WindowEvent { event, .. } => match event {

glutin::event::WindowEvent::CloseRequested => action = action::Action::Stop,

\_ => (),

},

\_ => (),

}

}

action

});

}

1. 实验结果截图

在配制好的环境下运行上述代码，可以得到以下结果

上面的线由 dda算法得到，下面的线和中间的圆由 Bresenham 算法得到。

1. 实验小结

通过本次实验，我掌握了几种常见图元的光栅化算法，以及 openGL 的基础使用，还有 glium 库的基本用法。