第4章图形几何变换上机作业

1. 实验题目

1. 利用OpenGL实现一个立方体关于参考点 (10.0,20.0,10.0) 进行放缩变换，放缩因子(2.0,1.0,0.5)。

2.利用OpenGL 实现一个矩形关于y=x+5对称的新图形。

3.通过定义键盘回调函数，每按一次空格键，让三个点依次完成画点、画线、画三角形，并让三角形沿三角形中心旋转起来。

1. 算法描述

1.立方体放缩变换

使用从网络上下载的立方体模型 cube.obj，用自行编写的 gl 库中的 vertex::load\_wavefront 函数加载为三角形，向 uniform 中传入变换矩阵 model。

[1.0 + t, 0.0, 0.0, 0.0],

[0.0, 1.0, 0.0, 0.0],

[0.0, 0.0, 1.0 - 0.5 \* t, 0.0],

[-10.0 \* t, 0.0, 5.0 \* t, 1.0f32],

其中 t = 1 时表明最终变换，t 从 0 到 1 表示变换过程。主对角线表示缩放倍率，第四行表示位移。

2.矩形对称变换

变换矩阵为

[0.0, 1.0, 0.0, 0.0],

[1.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 1.0, 0.0],

[-5.0 / w as f32, 5.0 / h as f32, 0.0, 1.0f32],

其中 w, h 为屏幕分辨率。

3. 三角形旋转变换

变换矩阵为

[t.sin(), t.cos(), 0.0, 0.0],

[t.cos(), -t.sin(), 0.0, 0.0],

[0.0, 0.0, 1.0, 0.0],

[0.0, 0.0, 0.0, 1.0f32],

其中 t 表示旋转角度

1. 绘图代码部分

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。gl 库中内置了一种点着色器，可以传入视角矩阵，透视矩阵和模型矩阵，本章主要通过传入模型矩阵来进行坐标变换。

以下是这个实验的所有源代码，也可以查看附件中 Chapter4/src/main.rs。

#[macro\_use]

extern crate glium;

use gl::camera;

use gl::shader;

use gl::action;

use gl::models;

use gl::vertex;

use gl::vertex::Vertex;

use glium::Display;

use glium::Surface;

use glium::glutin::event::ElementState;

pub fn draw\_cube\_stretch(display: &Display, t: f32) {

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

let camera = camera::CameraState::new();

let vertex\_buffer = models::cube(&display);

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

let params = glium::DrawParameters {

depth: glium::Depth {

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

write: true,

.. Default::default()

},

.. Default::default()

};

let uniforms = uniform! {

perspective: camera.get\_perspective(),

view: camera.get\_view(),

model: [

[1.0 + t, 0.0, 0.0, 0.0],

[0.0, 1.0, 0.0, 0.0],

[0.0, 0.0, 1.0 - 0.5 \* t, 0.0],

[-10.0 \* t, 0.0, 5.0 \* t, 1.0f32],

]

};

let mut target = display.draw();

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

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

target.finish().unwrap();

}

fn draw\_rectangle\_flip(display: &Display, rectangle: [i32; 4]) {

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

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

let vertex = vec![

Vertex::new\_2d(rectangle[0] as f32 / h as f32, rectangle[1] as f32 / w as f32),

Vertex::new\_2d(rectangle[2] as f32 / h as f32, rectangle[1] as f32 / w as f32),

Vertex::new\_2d(rectangle[2] as f32 / h as f32, rectangle[3] as f32 / w as f32),

Vertex::new\_2d(rectangle[0] as f32 / h as f32, rectangle[3] as f32 / w as f32),

];

let vertex\_buffer = vertex::from\_vertex(display, &vertex);

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

let params = glium::DrawParameters {

depth: glium::Depth {

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

write: true,

.. Default::default()

},

.. Default::default()

};

let uniforms = uniform! {

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

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

model: [

[0.0, 1.0, 0.0, 0.0],

[1.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 1.0, 0.0],

[-5.0 / w as f32, 5.0 / h as f32, 0.0, 1.0f32],

]

};

let mut target = display.draw();

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

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

target.finish().unwrap();

}

fn draw\_triangle\_step(display: &Display, step: i32, t : f32) {

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

let vertex = vec![

Vertex::new\_2d( -0.5, 0.5 ),

Vertex::new\_2d( 0.0, -0.5 ),

Vertex::new\_2d( 0.5, 0.5 ),

];

let vertex\_buffer = vertex::from\_vertex(display, &vertex);

let indices\_buffer = match step {

2 => glium::index::NoIndices(glium::index::PrimitiveType::Points),

3 => glium::index::NoIndices(glium::index::PrimitiveType::LineLoop),

\_ => glium::index::NoIndices(glium::index::PrimitiveType::TrianglesList),

};

let params = glium::DrawParameters {

depth: glium::Depth {

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

write: true,

.. Default::default()

},

.. Default::default()

};

let uniforms = uniform! {

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

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

model: [

[t.sin(), t.cos(), 0.0, 0.0],

[t.cos(), -t.sin(), 0.0, 0.0],

[0.0, 0.0, 1.0, 0.0],

[0.0, 0.0, 0.0, 1.0f32],

]

};

let mut target = display.draw();

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

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

target.finish().unwrap();

}

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 mut step = 0;

let mut t: f32 = 0.0;

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

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

if t < 1.0 { t += 0.001; } else { t = 0.0 };

match step {

0 => {

draw\_cube\_stretch(&display, t);

}

1 => draw\_rectangle\_flip(&display, [0, 100, 50, 0]),

\_ => draw\_triangle\_step(&display, step, t \* 6.28),

}

// 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; },

glutin::event::WindowEvent::KeyboardInput { device\_id: \_, input, is\_synthetic:\_ } =>

{ match input.state {

ElementState::Pressed => {step += 1;}

\_ => {}

} }

\_ => (),

},

\_ => (),

}

}

action

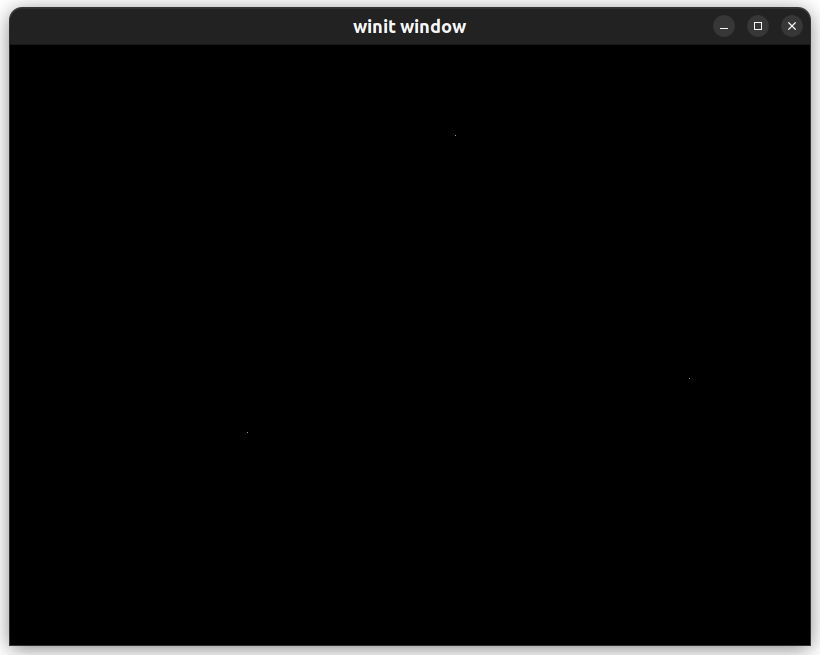
});

}

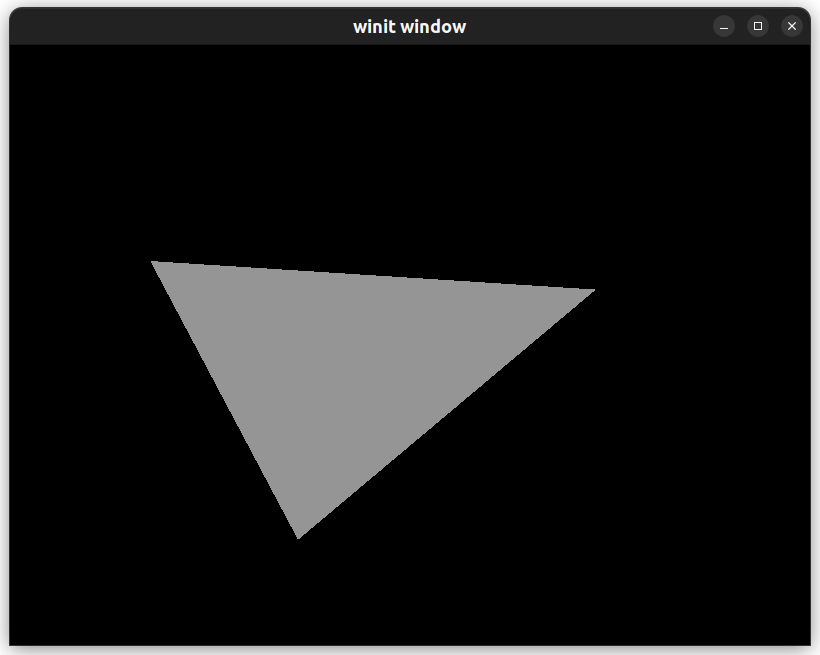
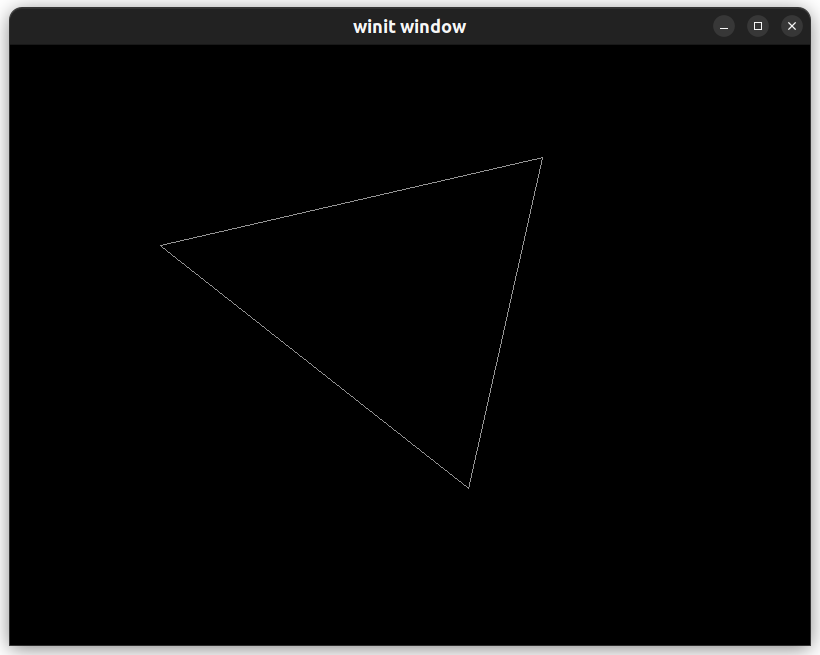
1. 实验结果截图

在配制好的环境下运行上述代码，每次按下空格可以得到一题的输出，具体如下。





可以看出三角形在正确旋转。



1. 实验小结

通过本次实验，我掌握了通过矩阵实现的坐标变换方法，以及着色器的基础使用。