第5章三维观察上机作业

1. 实验题目

1.编写程序，使一物体沿着一条直线匀速移动。

2.编写程序，使一物体围绕屏幕上一点匀速旋转。

1. 算法描述

1.三维物体匀速直线运动 & 匀速旋转

与上一章类似，只是增加了透视变换，在 gl::camera 中已经实现好了，矩阵为

[f / self.aspect\_ratio, 0.0, 0.0 , 0.0],

[ 0.0 , f , 0.0 , 0.0],

[ 0.0 , 0.0, (zfar+znear)/(zfar-znear) , 1.0],

[ 0.0 , 0.0, -(2.0\*zfar\*znear)/(zfar-znear), 0.0],

其中 aspect\_ratio 是屏幕长宽比，f 是误差率，zfar与 znear 为最远点与最近点。

传入点着色的 prespective 矩阵即可。

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。

以下是这个实验的所有源代码，也可以查看附件中 Chapter5/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;

fn matrix\_mutiple(x: [[f32; 4]; 4], y: [[f32; 4]; 4]) -> [[f32; 4]; 4] {

let mut ret = [

[0.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0f32],

];

for i in 0..4 {

for j in 0..4 {

for k in 0..4 {

ret[i][j] += x[i][k] \* y[k][j];

}

}

}

ret

}

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

let mut target = display.draw();

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

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, 0.0, 0.0, 0.0],

[0.0, 1.0, 0.0, 0.0],

[0.0, 0.0, 1.0, 0.0],

[-2.0, -1.5, 10.0 \* t -3.0, 1.0f32],

]

};

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

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

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

let vertex = vec![

Vertex::new\_3d\_point(-1.0, -0.5, -10.0),

Vertex::new\_3d\_point(-1.0, -0.5, 1000.0),

Vertex::new\_3d\_point(-3.0, -0.5, -10.0),

Vertex::new\_3d\_point(-3.0, -0.5, 1000.0),

Vertex::new\_3d\_point(-1.0, -2.5, -10.0),

Vertex::new\_3d\_point(-1.0, -2.5, 1000.0),

Vertex::new\_3d\_point(-3.0, -2.5, -10.0),

Vertex::new\_3d\_point(-3.0, -2.5, 1000.0),

];

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

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

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: camera::CameraState::element\_matrix()

};

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

target.finish().unwrap();

}

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

let mut target = display.draw();

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

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 t = t\*6.28;

let uniforms = uniform! {

perspective: camera.get\_perspective(),

view: camera.get\_view(),

model: matrix\_mutiple (

[

[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, -2.0, -2.0, 1.0],

],

[

[1.0\*t.sin(), 0.0, 1.0\*t.cos(), 0.0],

[0.0, 1.0, 0.0, 0.0],

[1.0\*t.cos(), 0.0, 1.0\*-t.sin(), 0.0],

[0.0, 0.0, 0.0, 1.0f32],

]

)

};

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

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

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

let vertex = vec![

Vertex::new\_3d\_point(0.0, -2.0, 0.0)

];

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

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

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: camera::CameraState::element\_matrix()

};

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\_move(&display, t);

}

\_ => {

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

}

}

// 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;}

\_ => {}

} }

\_ => (),

},

\_ => (),

}

}

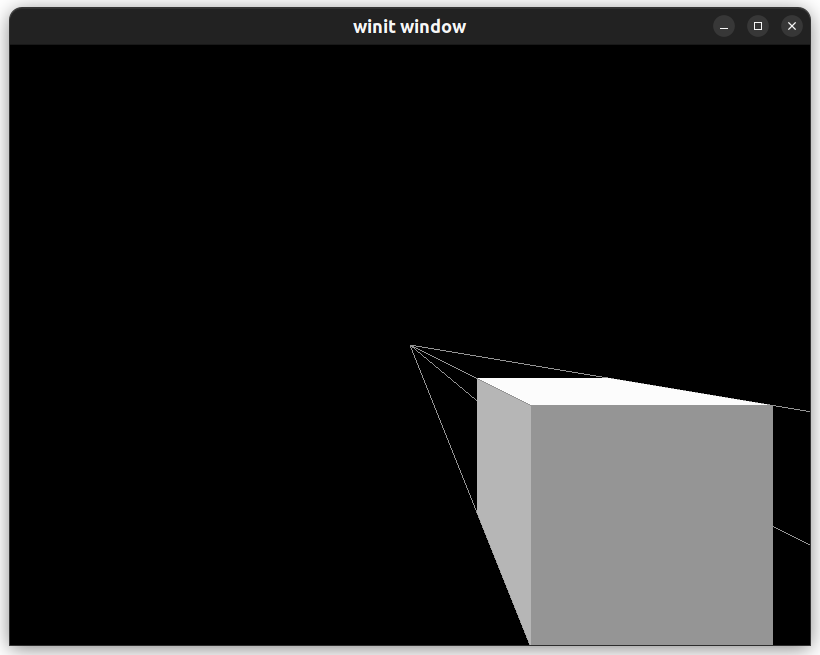
action

});

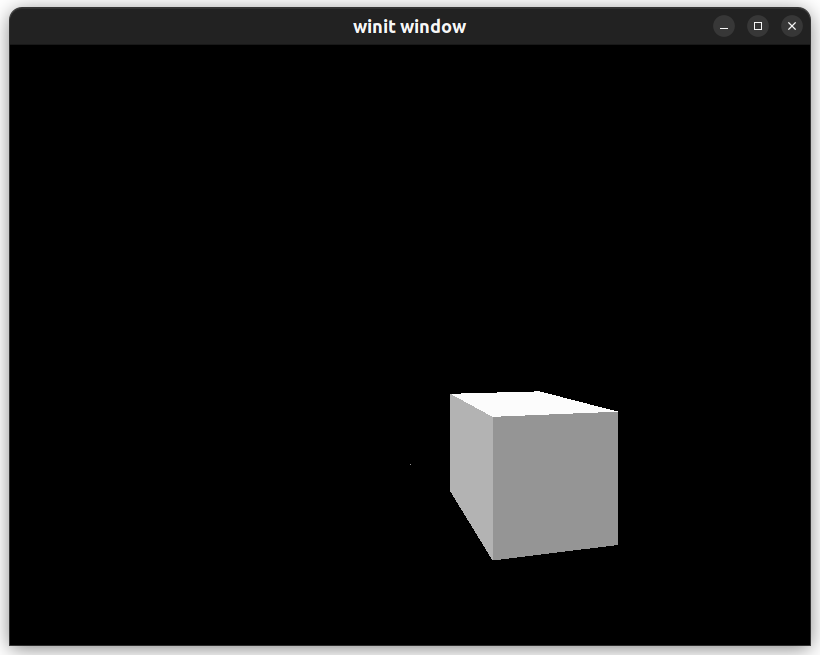
}

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。



立方体沿着四个顶点的线向无穷远处前进

立方体沿着屏幕中的小点旋转。

1. 实验小结

本次实验与第四章试验非常相似，增加了透视矩阵。