第6章曲线与曲面上机作业

1. 实验题目

1.证明如下的两条三次曲线段具有 C1 连续性，但没有 G1 连续性，并画出两条曲线段。

P1 = [t^2-2t+1, t^3-2t^2+t]

P2 = [t^2, t^3]

2.假 定 一 条 三 次 Hermite 曲 线 的 两 个 端 点 P1=<0,1>,P4=<3,0>, 端 点 处 切 向 量R1=<0,1>,R4=<-3,0>，请写出 Hermite 多项式形式，并绘出最后曲线，改变切向量，观  
 察曲线形状变化。

3.绘制曲线：已知 4 点 P1(0,0,0)、P2(1,1,1)、 P3(2,-1,-1)、P4(3,0,0)，用其作为控制点分别绘制 一条 Bezier 曲线、一条 B 样条曲线，并分别计算参数为 0、1/3、 2/3、1 时它们各自的位置矢 量。

4.绘制曲面：利用 Bezier 曲面构造茶壶的表面形状，定义控制点：  
 float ctrlpoints[4][4][3] = {  
 { {-2, -1, 0}, { -2.0, -1.0, 4.0},  
 { 2.0, -1.0, 4.0}, { 2, -1, 0} },  
 { {-3, 0, 0}, { -3.0, 0, 6.0},  
 { 3.0, 0, 6.0}, { 3, 0, 0}},  
 { {-1.5, 0.5, 0}, {-1.5, 0.5, 3.0},  
 {1.5, 0.5, 3.0}, {1.5, 0.5, 0}},  
 { {-2, 1, 0}, { -2.0, 1.0, 4.0},  
 { 2.0, 1.0, 4.0}, { 2, 1, 0} }  
 };

1. 算法描述

1.三次参数曲线

P1(t) = (0,1)t^3+(1,-2)t^2+(-2,1)t+1 P1`(1) = (0,3)t^2+(2,-4)t+(-2,1) = (0,0)

P2(t) = (0,1)t^3+(1,0)t^2 P2`(0) = (0,3)t^2+(2,0)t = (0,0)

所以 P1, P2 具有 C1 连续性

但是 P1 : dy/dx(1) = 2.5

P2 : dy/dx(0) = 0

所以 P1, P2 不具有 G1 连续性

绘图算法直接取很小的 dt 然后连线即可。

2. Hermite 曲 线

多项式形式为 P(t) = (-9,3)t^3+(12,-5)t^2+(0,1)t+(0,1)

绘制方法同上

3. Bezier 曲线

P(t)= C(3,0)P0(1-t)^3 + C(3,1)P1 t(1-t)^2 + C(3,2)P2 t^2(1-t)+ C(3,3)P3 t^3

同上模拟绘制即可。

4. Bezier曲面

先绘制四条贝塞尔曲线，然后用曲线上的点在同步绘制贝塞尔曲面即可。

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 Chapter6/src/main.rs。

#[macro\_use]

#[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::Surface;

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

pub fn draw\_parametric\_curve(display: &Display) {

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 mut vertex = Vec::new();

for i in 0..100 {

let t = i as f32 / 100.0;

vertex.push(Vertex::new\_2d(t \* t - 2.0 \* t + 1.0, t \* t \* t - 2.0 \* t \* t + t));

}

for i in 0..100 {

let t = i as f32 / 100.0;

vertex.push(Vertex::new\_2d(t \* t, t \* t \* t) );

}

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

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

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

};

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

target.finish().unwrap();

}

type Point = [f32; 2];

fn draw\_hermite\_curve(display: &Display, p1: Point, p2: Point, v1: Point, v2: Point) {

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 mut vertex = Vec::new();

vertex.push(Vertex::new\_2d(p1[0] + v1[0], p1[1] + v1[1]));

for i in 0..100 {

let t = i as f32 / 100.0;

let t2 = t \* t;

let t3 = t \* t2;

let exp = 3.0 \* t2 - 2.0 \* t3;

let x = (1.0 - exp) \* p1[0] + exp \* p2[0] + (t - 2.0 \* t2 + t3) \* v1[0] + (t3 - t2) \* v2[0];

let y = (1.0 - exp) \* p1[1] + exp \* p2[1] + (t - 2.0 \* t2 + t3) \* v1[1] + (t3 - t2) \* v2[1];

vertex.push(Vertex::new\_2d(x, y));

}

vertex.push(Vertex::new\_2d(p2[0] + v2[0], p2[1] + v2[1]));

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

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

let params = glium::DrawParameters {

depth: glium::Depth {

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

write: true,

.. Default::default()

},

.. Default::default()

};

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

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\_bezier\_curve(display: &Display) {

let p1 = [0.0, 0.0, 0.0f32];

let p2 = [1.0, 1.0, 1.0f32];

let p3 = [2.0, -1.0, -1.0f32];

let p4 = [3.0, 0.0, 0.0f32];

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 mut vertex = Vec::new();

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

let params = glium::DrawParameters {

depth: glium::Depth {

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

write: true,

.. Default::default()

},

.. Default::default()

};

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

camera.set\_position((0.0, 0.0, -3.0));

camera.set\_direction((0.0, 0.0, 1.0));

let uniforms = uniform! {

perspective: camera.get\_perspective(),

view: camera.get\_view(),

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

};

vertex.push(Vertex::new\_3d\_point(p1[0], p1[1], p1[2]));

vertex.push(Vertex::new\_3d\_point(p2[0], p2[1], p2[2]));

vertex.push(Vertex::new\_3d\_point(p3[0], p3[1], p3[2]));

vertex.push(Vertex::new\_3d\_point(p4[0], p4[1], p4[2]));

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

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

vertex.clear();

for i in 0..1000 {

let t = i as f32 / 1000.0;

let x = (1.0 - t).powi(3) \* p1[0] + 3.0 \* (1.0 - t).powi(2) \* t \* p2[0] + 3.0 \* (1.0 - t) \* t \* t \* p3[0] + t \* t \* t \* p4[0];

let y = (1.0 - t).powi(3) \* p1[1] + 3.0 \* (1.0 - t).powi(2) \* t \* p2[1] + 3.0 \* (1.0 - t) \* t \* t \* p3[1] + t \* t \* t \* p4[1];

let z = (1.0 - t).powi(3) \* p1[2] + 3.0 \* (1.0 - t).powi(2) \* t \* p2[2] + 3.0 \* (1.0 - t) \* t \* t \* p3[2] + t \* t \* t \* p4[2];

vertex.push(Vertex::new\_3d\_point(x, y, z));

}

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

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

target.finish().unwrap();

}

fn draw\_bezier\_surface(display: &Display) {

let p = [

[ [-2.0, -1.0, 0.0], [-2.0, -1.0, 4.0], [2.0, -1.0, 4.0], [2.0, -1.0, 0.0] ],

[ [-3.0, 0.0, 0.0], [-3.0, 0.0, 6.0], [3.0, 0.0, 6.0], [3.0, 0.0, 0.0] ],

[ [-1.5, 0.5, 0.0], [-1.5, 0.5, 3.0], [1.5, 0.5, 3.0], [1.5, 0.5, 0.0] ],

[ [-2.0, 1.0, 0.0], [-2.0, 1.0, 4.0], [2.0, 1.0, 4.0], [2.0, 1.0, 0.0f32] ]

];

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 mut vertex = Vec::new();

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

camera.set\_position((0.0, 0.0, 5.0));

camera.set\_direction((0.0, 0.0, -1.0));

let uniforms = uniform! {

perspective: camera.get\_perspective(),

view: camera.get\_view(),

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

};

for i in 0..1000 {

let t = i as f32 / 1000.0;

let mut pp = [[0.0; 3]; 4];

for x in 0..4 {

pp[x] = [

(1.0 - t).powi(3) \* p[x][0][0] + 3.0 \* (1.0 - t).powi(2) \* t \* p[x][1][0] + 3.0 \* (1.0 - t) \* t \* t \* p[x][2][0] + t \* t \* t \* p[x][3][0],

(1.0 - t).powi(3) \* p[x][0][1] + 3.0 \* (1.0 - t).powi(2) \* t \* p[x][1][1] + 3.0 \* (1.0 - t) \* t \* t \* p[x][2][1] + t \* t \* t \* p[x][3][1],

(1.0 - t).powi(3) \* p[x][0][2] + 3.0 \* (1.0 - t).powi(2) \* t \* p[x][1][2] + 3.0 \* (1.0 - t) \* t \* t \* p[x][2][2] + t \* t \* t \* p[x][3][2]

]

}

for j in 0..100 {

let t = j as f32 / 100.0;

let x = (1.0 - t).powi(3) \* pp[0][0] + 3.0 \* (1.0 - t).powi(2) \* t \* pp[1][0] + 3.0 \* (1.0 - t) \* t \* t \* pp[2][0] + t \* t \* t \* pp[3][0];

let y = (1.0 - t).powi(3) \* pp[0][1] + 3.0 \* (1.0 - t).powi(2) \* t \* pp[1][1] + 3.0 \* (1.0 - t) \* t \* t \* pp[2][1] + t \* t \* t \* pp[3][1];

let z = (1.0 - t).powi(3) \* pp[0][2] + 3.0 \* (1.0 - t).powi(2) \* t \* pp[1][2] + 3.0 \* (1.0 - t) \* t \* t \* pp[2][2] + t \* t \* t \* pp[3][2];

vertex.push(Vertex::new\_3d\_point(x, y, z));

}

}

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

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 += 0.01; } else { t = 0.0 };

match step {

0 => draw\_parametric\_curve(&display),

1 => draw\_hermite\_curve(&display, [0.0, 1.0], [3.0, 0.0], [-t, 1.0 - t], [-3.0 + t \* 3.0, t \* 1.0]),

2 => draw\_bezier\_curve(&display),

\_ => draw\_bezier\_surface(&display)

}

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

});

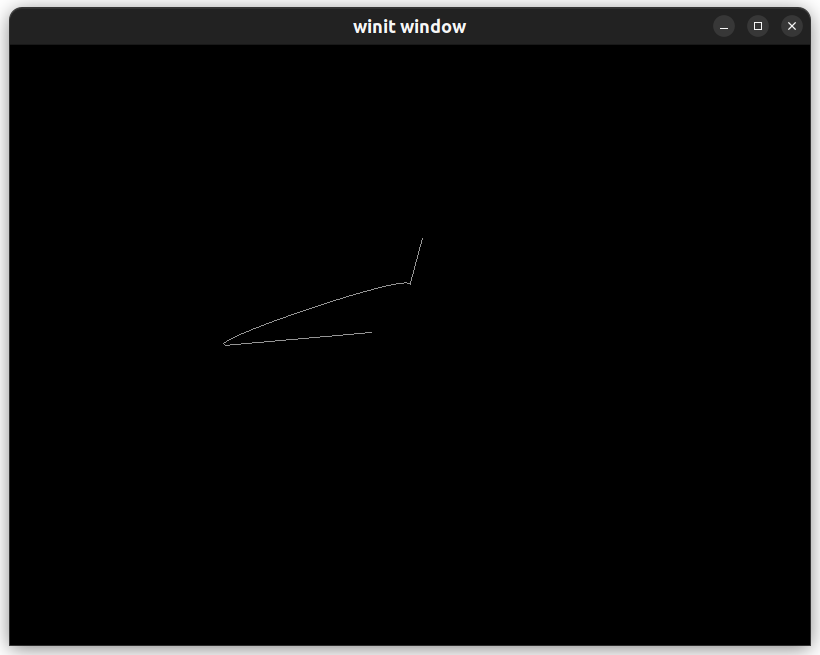
}

1. 实验结果截图

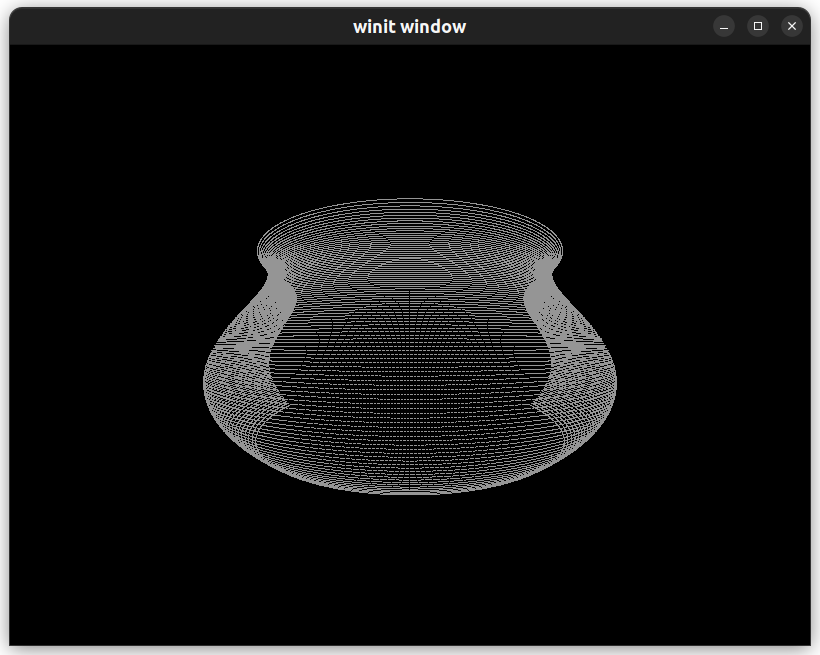
在配制好的环境下运行上述代码，每次按下空格可以得到一题的输出，具体如下。

参数曲线

Hermite 曲 线

Bezier 曲线

Bezier 曲面



1. 实验小结

通过本次实验我学习了计算机图形学中的曲线理论，并学会了分析连续性以及绘制方式。