­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Лабораторна робота №4

з курсу «Геометричне моделювання у конструюванні інженерних об'єктів та систем»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 3

Виконав студент гр. КНз-3

Чалий Михайло

­­

Львів 2015

## Мета роботи

Ознайомлення та практичне осовєння технологій й основ роботи з графічними можливостями бібліотеки OpenGL в операційному середовищі Windows. Вивчити сопособи підключення бібліотеки OpenGL. Ознаоймитись та засвоїти основні графічні функції та процедури, набути практичних навиків програмування графіки для Windows-програм на основі OpenGL.

## Теоретичні відомості

**OpenGL** — специфікація, що визначає незалежний від мови програмування крос-платформовий програмний інтерфейс (API) для написання застосунків, що використовують 2D та 3D комп'ютерну графіку. Цей інтерфейс містить понад 250 функцій, які можуть використовуватися для малювання складних тривимірних сцен з простих примітивів. Широко застосовується індустрією комп'ютерних ігор і віртуальної реальності, у графічних інтерфейсах (Compiz, Clutter), при візуалізації наукових даних, в системах автоматизованого проектування тощо.

### Специфікація

На базовому рівні, OpenGL — це всього лише специфікація, тобто, — просто документ, який описує набір функцій та їх точну поведінку. На основі цих специфікацій виробники апаратного забезпечення створюють реалізації — бібліотеки функцій, які відповідають заявленій в OpenGL специфікації. Ці реалізації проектуються для того, щоб при можливості використовувати можливості апаратного забезпечення. Коли апаратне прискорення не допускається, виконання функцій здійснюється за допомогою програмного забезпечення. Виробники повинні пройти спеціальні тести на відповідність, перш, ніж їхню реалізацію класифікуватимуть, як реалізацію OpenGL. Таким чином, розробникам програмного забезпечення необхідно всього лиш навчитися використовувати описані у специфікації функції, і лишити їхню реалізацію за розробниками апаратного забезпечення.

Ефективні реалізації OpenGL існують для операційних систем Linux, MacOS X, Microsoft Windows та багатьох UNIX-подібних ОС, а також для таких ігрових боксів, як Sony PlayStation 3. Різні програмні реалізації OpenGL існують для платформ, виробники яких не підтримують дану специфікацію. Відкрита (open source) бібліотека Mesa — повністю OpenGL сумісний програмний API. Однак, для того, щоб уникнути витрат на ліцензування, пов'язаних з формалізацією, яка вимагається для офіційного визнання реалізації, Mesa є неофіційною реалізацією специфікації, хоча й повністю з нею сумісна.

### Архітектура

OpenGL орієнтується на такі два завдання: Сховати складності адаптації різних 3D-прискорювачів, надаючи розробнику єдиний API. Приховати відмінності в можливостях апаратних платформ, вимагаючи реалізації відсутньої функціональності за допомогою програмної емуляції. Основним принципом роботи OpenGL є отримання наборів векторних графічних примітивів у вигляді точок, ліній та багатокутників з наступною математичною обробкою отриманих даних та побудовою растрової картинки на екрані і/або в пам'яті. Векторні трансформації та растеризация виконуються графічним конвеєром (graphics pipeline), який власне являє собою дискретний автомат. Абсолютна більшість команд OpenGL потрапляють в одну з двох груп: або вони додають графічні примітиви на вхід в конвеєр, або конфігурують конвеєр на різне виконання трансформацій. OpenGL є низькорівневим процедурним API, що змушує програміста диктувати точну послідовність кроків, щоб побудувати результуючу растрову графіку (імперативний підхід). Це є основною відмінністю від дескрипторних підходів, коли вся сцена передається у вигляді структури даних (найчастіше дерева), яке обробляється і будується на екрані. З одного боку, імперативний підхід вимагає від програміста глибокого знання законів тривимірної графіки та математичних моделей, з іншого боку — дає свободу впровадження різних інновацій.

## Завдання

3. Написати програму із застосуванням 2D графічних функцій бібліотеки OpenGL, яка дозволяє візуалізувати у чотирьох підпорядкованих віконних формах програми дитячий калейдоскоп з 8 осями симетрії та набором з 3 різнокольорових кристалів(багатокутники з різною кількістю кутів). Візерунок утворюється у результаті обертання калейдоскопа. Застосувати стандартні для OpenGL команди афінних перетворень.

## Реалізація

Лістінг 1. l4.js

/\* jshint esnext: true \*/

var React = require('react');

var assert = require('assert');

var mat4 = require('gl-matrix').mat4;

require('es6-shim');

var ReactBootstrap = require('react-bootstrap');

var Input = ReactBootstrap.Input;

function rand(min, max) {

return Math.floor(Math.random() \* (max - min)) + min;

}

function randf(min, max) {

return Math.random() \* (max - min) + min;

}

function rad(degrees) {

return degrees \* Math.PI / 180;

}

function compileShader(gl, type, src){

var shader = gl.createShader(type);

gl.shaderSource(shader, src);

gl.compileShader(shader);

if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS)) {

console.log(gl.getShaderInfoLog(shader));

return null;

}

return shader;

}

function prepareArrayBuffer(gl, size, vertices){

var buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, buffer);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STATIC\_DRAW);

buffer.itemSize = size;

buffer.numItems = vertices.length / size;

return buffer;

}

function prepareElementArrayBuffer(gl, size, vertices){

var buffer = gl.createBuffer();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, buffer);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(vertices), gl.STATIC\_DRAW);

buffer.itemSize = size;

buffer.numItems = vertices.length / size;

return buffer;

}

function prepareTexture(gl, src){

var texture = gl.createTexture();

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

new Uint8Array([255, 255, 255, 255]));

texture.image = new Image();

texture.image.onload = function () {

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.pixelStorei(gl.UNPACK\_FLIP\_Y\_WEBGL, true);

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, texture.image);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.NEAREST);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.NEAREST);

gl.bindTexture(gl.TEXTURE\_2D, null);

};

texture.image.src = src;

return texture;

}

function prepareColorTexture(gl, color){

var texture = gl.createTexture();

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

new Uint8Array(color));

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

return texture;

}

function buildCorneredFigure(gl){

var centerZ = 0.1;

var arrisZ = 0.1;

var lowerZ = 0.1;

var num = rand(1,4) \* 4;

var x = randf(-1.0, 5.0);

var y = randf(-1.0, 5.0);

var size = randf(0.5, 1.5);

var size2 = randf(size/4, size\*0.7);

var color = [rand(1, 255), rand(1, 255), rand(1, 255), rand(126, 255)];

var vertices = [];

var indices = [];

for (var i = 0; i <= num; i++) {

var ang = i \* 2 \* Math.PI / num;

vertices.push(x, y, centerZ);

vertices.push(

x + size \* Math.cos(ang),

y + size \* Math.sin(ang),

arrisZ

);

var ang2 = ((i \* 2) + 1) \* Math.PI / num;

vertices.push(

x + size2 \* Math.cos(ang2),

y + size2 \* Math.sin(ang2),

lowerZ

);

vertices.push(x, y, centerZ);

vertices.push(

x + size2 \* Math.cos(ang2),

y + size2 \* Math.sin(ang2),

lowerZ

);

var ang3 = ((i \* 2) + 2) \* Math.PI / num;

vertices.push(

x + size \* Math.cos(ang3),

y + size \* Math.sin(ang3),

arrisZ

);

}

var textureCoords = Array.from({length: (vertices.length / 3) \* 2}, (v, k) => 1.0);

return {

mode: gl.TRIANGLES,

positions: prepareArrayBuffer(gl, 3, vertices),

textureCoords: prepareArrayBuffer(gl, 2, textureCoords),

texture: prepareColorTexture(gl, color),

x: x,

y: y

};

}

var L4 = React.createClass({

getInitialState: function(){

return {

angle: 10

};

},

initCanvas: function(){

if (this.refs.canvas){

var canvas = this.refs.canvas.getDOMNode();

var gl = this.gl = canvas.getContext("webgl");

gl.viewportWidth = canvas.width;

gl.viewportHeight = canvas.height;

gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);

var mvMatrix = this.mvMatrix = mat4.create();

var pMatrix = this.pMatrix = mat4.create();

// Init shaders

var fragmentShader = compileShader(gl, gl.FRAGMENT\_SHADER, `

precision mediump float;

varying vec2 vTextureCoord;

uniform sampler2D uSampler;

void main(void) {

gl\_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));

}

`);

var vertexShader = compileShader(gl, gl.VERTEX\_SHADER, `

attribute vec3 aVertexPosition;

attribute vec2 aTextureCoord;

uniform mat4 uMVMatrix;

uniform mat4 uPMatrix;

varying vec2 vTextureCoord;

void main(void) {

gl\_Position = uPMatrix \* uMVMatrix \* vec4(aVertexPosition, 1.0);

vTextureCoord = aTextureCoord;

}

`);

var shaderProgram = this.shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertexShader);

gl.attachShader(shaderProgram, fragmentShader);

gl.linkProgram(shaderProgram);

if (!gl.getProgramParameter(shaderProgram, gl.LINK\_STATUS)) {

console.log("Could not initialise shaders");

}

gl.useProgram(shaderProgram);

shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");

gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");

gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");

shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");

shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");

// Init figures

this.figures = [{

mode: gl.TRIANGLES,

positions: prepareArrayBuffer(gl, 3, [

3.0, 3.0, 0.0,

1.0, 3.0, 0.0,

3.0, 1.0, 0.0,

1.0, 1.0, 0.0

]),

textureCoords: prepareArrayBuffer(gl, 2, [

1.0, 1.0,

0.0, 1.0,

1.0, 0.0,

0.0, 0.0

]),

texture: prepareTexture(gl, 'assets/geom.png'),

indices: prepareElementArrayBuffer(gl, 1, [

0, 1, 2, 2, 3, 1

])

},

buildCorneredFigure(gl),

buildCorneredFigure(gl),

buildCorneredFigure(gl)];

// Clenup

gl.clearColor(1.0, 1.0, 1.0, 1.0);

gl.enable(gl.DEPTH\_TEST);

}

},

mvMatrixStack: [],

\_pushMvMatrix: function () {

var copy = mat4.create();

mat4.copy(copy, this.mvMatrix);

this.mvMatrixStack.push(copy);

},

\_popMvMatrix: function() {

if (this.mvMatrixStack.length === 0) {

throw "Invalid popMatrix!";

}

this.mvMatrix = this.mvMatrixStack.pop();

},

renderFigure: function(f){

assert(this.gl, 'GL context should be initialized');

var gl = this.gl;

var shaderProgram = this.shaderProgram;

gl.bindBuffer(gl.ARRAY\_BUFFER, f.positions);

gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, f.positions.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY\_BUFFER, f.textureCoords);

gl.vertexAttribPointer(shaderProgram.textureCoordAttribute, f.textureCoords.itemSize, gl.FLOAT, false, 0, 0);

gl.activeTexture(gl.TEXTURE0);

gl.bindTexture(gl.TEXTURE\_2D, f.texture);

gl.uniform1i(shaderProgram.samplerUniform, 0);

gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, this.pMatrix);

gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, this.mvMatrix);

if (f.indices){

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, f.indices);

gl.drawElements(f.mode, f.indices.numItems, gl.UNSIGNED\_SHORT, 0);

} else {

gl.drawArrays(f.mode, 0, f.positions.numItems);

}

},

renderCanvas: function(){

var gl = this.gl;

var renderBuffer = this.renderBuffer;

var shaderProgram = this.shaderProgram;

gl.viewport(0, 0, this.width, this.height);

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

mat4.perspective(this.pMatrix, 45.0, this.width / this.height, 0.1, 100.0);

for(let f of this.figures){

mat4.identity(this.mvMatrix);

mat4.translate(this.mvMatrix, this.mvMatrix, [0.0, 0.0, -7.0]);

// mat4.rotate(this.mvMatrix, this.mvMatrix, rad(45), [1, 0, 0]);

// mat4.rotate(this.mvMatrix, this.mvMatrix, rad(45), [0, 1, 0]);

// mat4.rotate(this.mvMatrix, this.mvMatrix, rad(45), [0, 0, 1]);

var num = 16;

mat4.rotate(this.mvMatrix, this.mvMatrix, rad(this.state.angle), [0, 0, 1]);

for(var i = 0; i < num; i++){

mat4.rotate(this.mvMatrix, this.mvMatrix, rad(360/num), [0, 0, 1]);

this.\_pushMvMatrix();

if (i%2 === 0){

mat4.scale(this.mvMatrix, this.mvMatrix, [-1.0, 1.0, 1.0]);

}

this.renderFigure(f);

this.\_popMvMatrix();

}

}

},

queueRenderCanvas: function(){

var self = this;

requestAnimationFrame(function(){

self.renderCanvas();

});

},

componentDidMount: function() {

var canvas = this.refs.canvas.getDOMNode();

this.width = canvas.width;

this.height = canvas.height;

canvas.addEventListener("mousedown", this.handleMouseClick, false);

this.initCanvas();

this.renderCanvas();

var self = this;

// Hack to ensure textures loaded

window.setTimeout(this.queueRenderCanvas, 500);

},

handleMouseClick: function(e){

var x = e.x;

var y = e.y;

var canvas = this.refs.canvas.getDOMNode();

x -= canvas.offsetLeft;

y -= canvas.offsetTop;

// Do something when click

},

componentWillUpdate: function(nextProps, nextState) {

this.queueRenderCanvas();

},

render: function() {

return <div className='container'>

<div className='row'>

<div className='col-md-8'><canvas ref='canvas' width={600} height={400} style={{borderColor: 'gray', borderThickness: '1', borderStyle: 'solid'}} ></canvas></div>

<div className='col-md-4'>

<form>

<Input name='angle' type='range' label='Кут' onChange={this.\_handleChange} />

</form>

</div>

</div>

</div>;

},

\_handleChange: function(e){

this.setState({

angle: (45 \* 100) / Math.max(5, e.target.value)

});

}

});

exports.L4 = L4;

Повна версія коду доступна на https://github.com/chaliy/studies-octo-adventure/tree/master/lp/c3\_2/gm/src/

## Результат

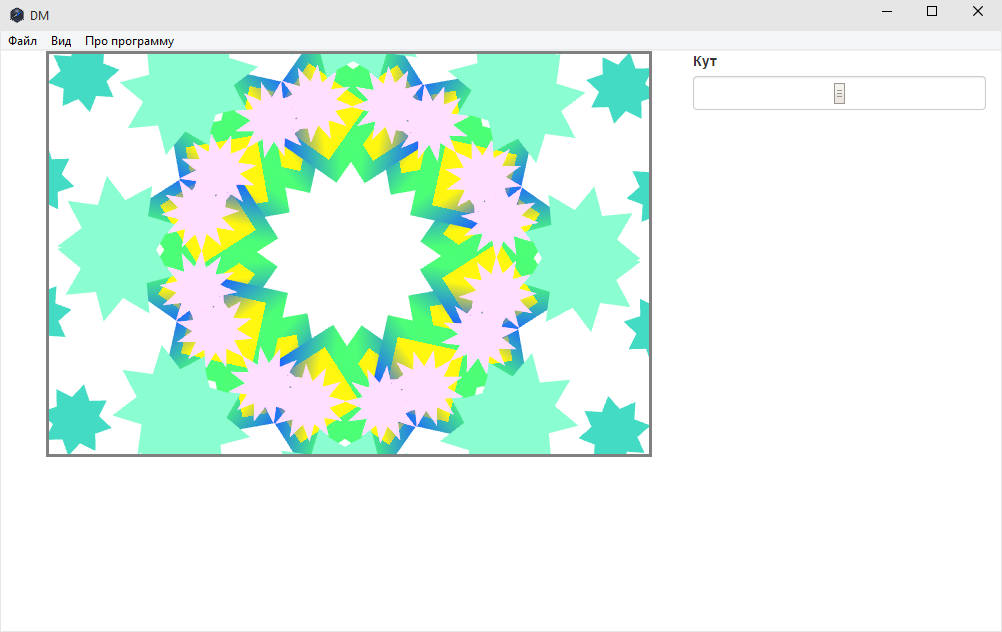


Рис 1. Перший етап

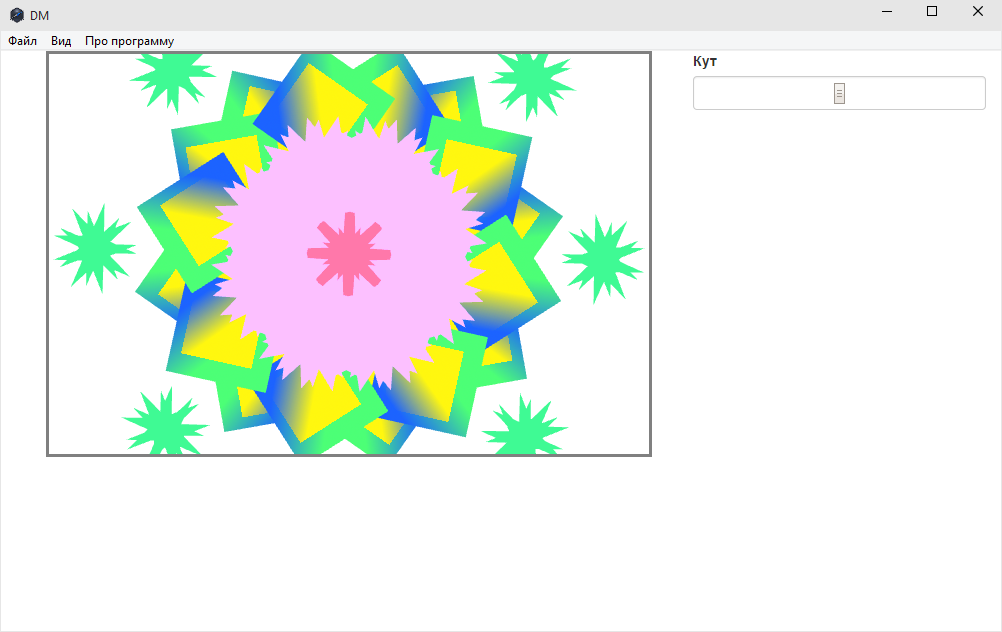


Рис 2. Другий етап

Програма також доступна на http://chaliy.name/play/lp/c3\_2/gm/#/l4

## Висновки

Ознайомився та практично засвоїв технології й основи роботи з графічними можливостями бібліотеки OpenGL в операційному середовищі Windows. Вивчив сопособи підключення бібліотеки OpenGL. Ознаоймився та засвоїв основні графічні функції та процедури, набув практичних навиків програмування графіки для Windows-програм на основі OpenGL.