TASK

Draw A Circle by using open GL.

Color the circle also.

#include <string>

#include <sstream>

#include <iostream>

#include <vector>

#include <GL\glew.h>

#include <GL\freeglut.h> // we use glut here as window manager

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

class Renderer {

public:

float t;

int mode;

private:

GLuint bufID;

int bufSize;

public:

// constructor

Renderer() : t(0.0), mode(0), bufID(0), bufSize(0) {}

//destructor

~Renderer() {

if (bufID != 0) glDeleteBuffers(1, &bufID);

}

public:

void init() {

glEnable(GL\_DEPTH\_TEST);

// generating VBO input data

std::vector<float> dataIn;

unsigned ayimutSegs = 1000;

unsigned polarSegs = 1000;

float ayimutStep = 2.0f \* float(M\_PI) / float(ayimutSegs);

float polarStep = float(M\_PI) / float(polarSegs);

float r = 1.0f;

bufSize = 0;

for (unsigned m = 0; m < ayimutSegs; m++) {

for (unsigned n = 0; n < polarSegs; n++) {

float phi = ayimutStep \* m;

float theta = polarStep \* n;

// compute xyz from spherical coordinates

float x = r \* sin(theta) \* cos(phi);

float y = r \* sin(theta) \* sin(phi);

float z = r \* cos(theta);

dataIn.push\_back(x);

dataIn.push\_back(y);

dataIn.push\_back(z);

bufSize++;

}

}

// generating VBO

glGenBuffers(1, &bufID);

glBindBuffer(GL\_ARRAY\_BUFFER, bufID);

glBufferData(GL\_ARRAY\_BUFFER, dataIn.size() \* sizeof(float), &dataIn[0], GL\_STATIC\_DRAW);

}

void resize(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(30.0, (float)w / (float)h, 0.1, 50.0);

}

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// set camera

gluLookAt(3.5, -1.0, 3.5, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);

// draw scene

glRotatef(t, 0.0f, 0.0f, 1.0f);

// activating VBO

glBindBuffer(GL\_ARRAY\_BUFFER, bufID);

int stride = 0;

glVertexPointer(3, GL\_FLOAT, stride, NULL);

glEnableClientState(GL\_VERTEX\_ARRAY);

if (mode == 0) {

glColor3f(0.0f, 1.0f, 0.0f);

}

else {

glColorPointer(3, GL\_FLOAT, stride, NULL);

glEnableClientState(GL\_COLOR\_ARRAY);

}

// render VBO

glDrawArrays(GL\_POINTS, 0, bufSize);

glDisableClientState(GL\_VERTEX\_ARRAY);

glDisableClientState(GL\_COLOR\_ARRAY);

}

};

//this is a static pointer to a Renderer used in the glut callback functions

static Renderer\* renderer;

//glut static callbacks start

static void glutResize(int w, int h)

{

renderer->resize(w, h);

}

static void glutDisplay()

{

renderer->display();

glutSwapBuffers();

glutReportErrors();

}

static void timer(int v)

{

float offset = 0.25f;

renderer->t += offset;

glutDisplay();

glutTimerFunc(unsigned(20), timer, ++v);

}

static void glutKeyboard(unsigned char key, int x, int y) {

bool redraw = false;

std::string modeStr;

switch (key) {

case '1':

if (renderer->mode == 1) renderer->mode = 0;

else renderer->mode = 1;

redraw = true;

break;

}

if (redraw) {

glutDisplay();

}

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(100, 100);

glutInitWindowSize(800, 600);

glutCreateWindow("EP1849134 (ZAINAB ASIF) BSCS-III SEC(A)");

GLenum err = glewInit();

if (GLEW\_OK != err) {

fprintf(stderr, "Glew error: %s\n", glewGetErrorString(err));

}

glutDisplayFunc(glutDisplay);

glutIdleFunc(glutDisplay);

glutReshapeFunc(glutResize);

glutKeyboardFunc(glutKeyboard);

renderer = new Renderer;

renderer->init();

glutTimerFunc(unsigned(20), timer, 0);

glutMainLoop();

}

OUTPUT

