电子科技大学

计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 计算机图形学

电子科技大学教务处制表

电 子 科 技 大 学

实　验　报　告

学生姓名：宋邦睿　　学 号：2023110903001　　指导教师：肖逸飞

实验地点：主楼 A2 413-2　　实验时间：7-10 周周六

一、实验项目名称：计算机图形学

二、实验学时：　10 学时

三、实验原理：

1、贝塞尔曲线

a):Bezier 绘制：根据贝塞尔函数的定义绘制控制点

用户操作：通过 glutMouseFunc(mouseButton); glutMotionFunc(mouseMotion);glutKeyboardFunc(keyboard);三个函数监听用户输入。通过 unProject 函数接口从屏幕坐标系逆映射回世界坐标系。实现了移动控制点、自定义控制点位置等操作。

b):多 Bezier 绘制：通过 GLWord 类管理所有控制点，实现了绘制多条 Bezier 曲线。

c):自动化移动控制点：自定义 AutoMove（）函数，配合 OpenGL 的库函数 glutTimerFunc（）实现程序重复自动调用 AutoMove（）随机改变选中控制点的位置，实现控制点自动移动。为使多控制点自动移动，设置了控制点储存容器储存选中控制点，同时被 AutoMove（）调用

2、分形几何

A）：通过迭代实现分形

B）：设置视口个数：通过 glViewport 函数建立四个视口，每个视口独立迭代

C）：自动迭代：glutTimerFunc（）实现自动迭代

四、实验内容及步骤：

1、Bezier 曲线

a):设置基类 GLObject 和 GLWorld 管理所有控制点的生成、删除、绘制。绘制时调用派生类自己的绘制函数。体现多态特性。

GLObject 类如下

```
class GLObject
{
public:
    GLint id = 0;
    GLColor color;
    GLTransform transform;
    bool movable = false;
    bool visible = true;

    GLWorld* parentWorld;

    virtual void Draw(GLenum RenderMode = GL_RENDER) {}

    virtual void MakePoint(GLdouble x, GLdouble y, GLdouble

    virtual void KeyboardMove(GLdouble right, GLdouble up) {

    virtual void AutoMove() {}

    void SetPosition(GLdouble x, GLdouble y, GLdouble z) {
        transform.x = x;
        transform.y = y;
        transform.z = z;
    }
};
```

Draw 由子类重写，用于绘制

MakePoint 由子类重写，内部调用 GLWorld 的 NewObject 方法创建对象

AutoMove 由子类重写，负责程序自动完成控制点变化

SetPosition 用于设置物体位置

GLWorld 类如下

```
class GLWorld
{
public:
    GLObject * objects[MAX_OBJECTS];
    GLint count = 0;

    GLWorld() = default;
    ~GLWorld() {
        for (int i = count - 1; i >= 0; --i)
        {
            delete objects[i];
        }

    }

    template < typename T>
    T * NewObject() {
        if (count < MAX_OBJECTS)
        {
            T * ret = new T;
            objects[count] = (GLObject*)ret;
            objects[count]->parentWorld = this;
            objects[count]->id = count;
            ++count;
            return ret;
        }
        else return nullptr;

    }
```

```
void DeleteObject(GLint id) {
    delete objects[id];
    for (int i = id; i < count - 1; i++)
    {
        objects[i] = objects[i + 1];
    }
    count -= 1;
}
void Empty() {
    for (int i = 0; i < count; i++)
    {
        delete objects[i];
    }
    count = 0;
}

void DrawObjects(GLenum RenderMode = GL_RENDER) {
    for (int i = 0; i < count; ++i)
    {
        objects[i]->Draw(RenderMode);
    }
}

std::set<GLint> id;
GLint pickedObjectId = -1;
void PickObject(GLint xMouse, GLint yMouse) {
    pickRects(GLUT_LEFT_BUTTON, GLUT_DOWN, xMouse, yMouse);

}
```

```
private:
    void pickRects(GLint button, GLint action, GLint xMouse, GLint yMouse);
    void processPicks(GLint nPicks, GLuint pickBuffer[]);
};
```

DeleteObject 和 Empty 用于清空指定对象和重置所有对象。

DrawObject 调用派生类 **Draw** 方法进行绘制

B）：对象的实例化

对象的实例化依靠 GLWorld 的 NewObjec 方法，根据传入对象的指针类型返回实例化对象指针，并记录在列表 object[]中统一管理。

```
template < typename T>
T* NewObject() {
    if (count < MAX_OBJECTS)
    {
        T* ret = new T;
        objects[count] = (GLObject*)ret;
        objects[count]->parentWorld = this;
        objects[count]->id = count;
        ++count;
        return ret;
    }
    else return nullptr;
```

C）：对象拾取功能

pickRects 实现对象拾取功能

button 和 action: 指示鼠标按键和动作（例如左键按下）。

xMouse, yMouse: 鼠标点击的位置（屏幕坐标）。

定义选择缓冲区 pickBuffer，用来存储拾取信息（如对象名称和深度信息）。

glSelectBuffer 告诉 OpenGL 使用指定的缓冲区记录拾取数据。

```
GLuint pickBuffer[DEFAULT_PICK_BUFFER_SIZE];
glSelectBuffer(DEFAULT_PICK_BUFFER_SIZE, pickBuffer);
```

glInitNames 清空命名栈（名字栈）。

glPushName(MAX_OBJECTS) 向命名栈推入一个值，作为初始栈顶值。

这个栈用于标识绘制的对象，后续可以使用 glLoadName(id) 替换栈顶元素，标识具体的对象。

```
glInitNames();
glPushName(MAX_OBJECTS);
```

切换到投影矩阵，保存当前矩阵状态。

设置拾取区域：使用 gluPickMatrix 创建一个以鼠标点击为中心的小矩形区域作为拾取区域，大小由 DEFAULT_PICK_WINDOW_SIZE 决定。

调用 gluOrtho2D 设置拾取区域的正交投影，以匹配应用的世界坐标范围。

调用 glRenderMode(GL_RENDER) 结束选择模式，返回拾取到的对象数 nPicks。

缓冲区中存储了拾取到的对象信息（如名称、深度等）。

调用 processPicks 函数对拾取到的对象进行处理。

对 **processPicks** 的进一步解释另作分析。

```
void GLWorld::pickRects(GLint button, GLint action, GLint xMouse,
    GLint yMouse)
{
    GLuint pickBuffer[DEFAULT_PICK_BUFFER_SIZE];
    GLint nPicks, vpArray[4];

    if(button != GLUT_LEFT_BUTTON || action != GLUT_DOWN)
        return;

    glSelectBuffer(DEFAULT_PICK_BUFFER_SIZE, pickBuffer); // Designate pick buffer.

    glRenderMode(GL_SELECT); // Activate picking operations.

    glInitNames(); // Initialize the object-ID stack.


    // 只使用一个栈顶元素，后续为图形命名使用 glLoadName(id) 替换栈顶

    glPushName(MAX_OBJECTS);
```

```
/* Save current viewing matrix. */
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();

glGetIntegerv(GL_VIEWPORT, vpArray);
gluPickMatrix(GLdouble(xMouse), GLdouble(vpArray[3] - yMouse),DEFAULT_PICK_WINDOW_SIZE, DEFAULT_PICK_WI

gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
//rects (GL_SELECT); // Process the rectangles in selection mode.
this->DrawObjects(GL_SELECT);
/* Restore original viewing matrix. */
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glFlush();
    /* Determine the number of picked objects and return to the
    * normal rendering mode.
    */
nPicks = glRenderMode(GL_RENDER);
processPicks(nPicks, pickBuffer); // Process picked objects.


glutPostRedisplay();
}
```

D）：GLPoint 类实现

GLPoint 类作为 Bezier 的基类，需要实现对应 Draw 方法、基础的位置计算、变换方法：

```
class GLPoint : public GLObject
{
public:
    GLPoint() {
        color.r = 1;
    }

    virtual void KeyboardMove(GLdouble right, GLdouble up) override{ ... }
    virtual void AutoMove() override {
        // 创建随机数生成器
        std::random_device rd;                    // 随机设备，提供种子
        std::mt19937 gen(rd());                   // Mersenne Twister 生成器
        std::uniform_real_distribution<double> dis(-10.0, 10.0); // [0, 10) 均匀分布

        // 生成随机数
        GLdouble a[2];
        for (int i = 0; i < 2; i++){ ... }
        KeyboardMove(a[0], a[1]);

    }

    virtual void Draw(GLenum RenderMode = GL_RENDER){ ... }

};
```

KeyboardMove 用于响应用户操作，具体实现见下文

AutoMove 用于执行程序自动控制控制点移动

Draw 作为绘制方法，实现为：

```
virtual void Draw(GLenum RenderMode = GL_RENDER) {
    // 需要在绘制前调用 SetPostion, 否则绘制在原点
    if (!visible) return;
    if (RenderMode == GL_SELECT) glLoadName((GLuint)id);
    glColor3f(color.r, color.g, color.b);
    glPointSize(DEFAULT_POINT_SIZE);
    glBegin(GL_POINTS);
    glVertex3d(transform.x, transform.y, transform.z);
    glEnd();
}
```

E）：Bezier 曲线计算

由书中给出的计算公式，实现相应计算部分。

```
void bezier(wcPt3D* ctrlPts, GLint nCtrlPts, GLint nBezCurvePts) {
    wcPt3D bezCurvePt;
    GLfloat u;
    GLint* C, k;
    C = new GLint[nCtrlPts];
    binomialCoeffs(nCtrlPts - 1, C);
    for (k = 0; k <= nBezCurvePts; k++)
    {
        u = GLfloat(k) / GLfloat(nBezCurvePts);
        computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
        plotPoint(bezCurvePt);
        //printf("%d/t", k);
    }
    delete[] C;
}
```

```
void bezier(GLPoint* ctrlPts[], GLint nBezCurvePts) {
    // 函数重载转发
    wcPt3D innerCtrlPts[MAX_BEZIER_CONTROL_POINTS_NUM];
    for (int i = 0; i < count; ++i)
    {
        innerCtrlPts[i].x = (GLfloat)ctrlPts[i]->transform.x;
        innerCtrlPts[i].y = (GLfloat)ctrlPts[i]->transform.y;
        innerCtrlPts[i].z = (GLfloat)ctrlPts[i]->transform.z;
    }

    bezier(innerCtrlPts, count, nBezCurvePts);


    // 绘制控制点连线
    if (ctrlPts[0] && ctrlPts[0]->visible == false) return;
    glColor3f(0.0, 1.0, 0.0);
    glLineWidth(4.0);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < count; ++i)
    {
        glVertex3d(ctrlPts[i]->transform.x, ctrlPts[i]->transform.y, ctrlPts[i]->transform.z);
    }
    glEnd();
}
```

D）：响应用户操作

unProject 函数响应用户鼠标点击位置逆映射计算世界坐标系位置，实现自定义控制点生成。

```
void unProject(GLdouble winx, GLdouble winy, GLdouble winz, GLdouble * objx,
    GLdouble * objy, GLdouble * objz)
    {
        // 从窗口到世界坐标系
        GLint viewport[4];
        GLdouble modelview[16];
        GLdouble projection[16];
        glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
        glGetDoublev(GL_PROJECTION_MATRIX, projection);
        glGetIntegerv(GL_VIEWPORT, viewport);
        gluUnProject(winx, winy, winz, modelview, projection, viewport, objx, objy, objz);
}
```

添加了 KeyboardMove 方法，便于用户用方向键控制控制点移动。同时给 AutoMove 函数提供接口，实现程序自动控制控制点移动。KeyboardMove 函数增加了越界判断，防止控制点离开屏幕坐标系。

```
virtual void KeyboardMove(GLdouble right, GLdouble up) override
{
    if (transform.x + right > xwcMax || transform.x + right < xwcMin) {
        if (transform.x + right > xwcMax)
            transform.x = right - xwcMax;
        else
            transform.x = xwcMax + right;
        return;
    }
    transform.x += right;
    if (transform.y + up > ywcMax || transform.y + up < ywcMin) {
        if (transform.y + up > ywcMax)
            transform.y = up - ywcMax;
        else
            transform.y = up + ywcMax;
        return;
    }
    transform.y += up;
}
```

E）：程序自动控制控制点位置的实现

增加 AutoMove 方法配合 OpenGl 库函数 glutTimerFunc（600, Auto, 0）每隔 1s 自动调用实例对象的 AutoMove 方法实现控制点自动移动。其中 Auto 函数设置在主函数中，调用 GLWorld 中选中的控制点，实现多控制点的同时自动化控制。

为了更好的控制多个控制点，定义集合容器 set 存储选中的控制点。processPicks 函数中将鼠标选中的控制点添加入 set 容器 id 中，id 将管理所有选中的控制点指针，便于自动控制。

```
void GLWorld::processPicks(GLint nPicks, GLuint pickBuffer[])
{

    // 储存获取到的最后一个 ID，即认为后绘制的图像在先绘制的图形之上

    if (nPicks == 0) pickedObjectId = -1;
    else
    {

        // 因为只使用一个栈顶元素，所以可以由公式计算得出最后一个图形ID
        int chunkId = 4 * (nPicks - 1);
        pickedObjectId = pickBuffer[chunkId + 3];
        id.insert(pickedObjectId);
    }
    return;
}
```

Auto 函数的实现：

```
void Auto(int value) {
    if (MyWorld.id.empty())
        return;
    for (auto ite = MyWorld.id.begin(); ite != MyWorld.id.end(); ite++)
        MyWorld.objects[*ite]->AutoMove();
    }

    glutPostRedisplay();
    glutTimerFunc(600, Auto, 0); // ~1000ms = 1s
}
```

AutoMove 函数的实现：

```
virtual void AutoMove() override {
    // 创建随机数生成器
    std::random_device rd;                          // 随机设备，提供种子
    std::mt19937 gen(rd());                         // Mersenne Twister 生成器
    std::uniform_real_distribution<double> dis(-10.0, 10.0); // [0, 10) 均匀分布

    // 生成随机数
    GLdouble a[2];
    for (int i = 0; i < 2; i++)
    {
        a[i] = GLdouble(dis(gen));
    }
    KeyboardMove(a[0], a[1]);

}
```

综上，实现生成贝塞尔曲线，测试如下：

F）主函数部分

设置菜单绑定鼠标滚轮，菜单映射 init 部分

```
static menuEntryStruct mainMenu[] = {
"Reset", '0',
"Create Bezier/Finish Bezier", '1',
"Quit", 27, //ESC 键（ASCII: 27）
"Auto Moving",'2',//Automoving
//"MovePoint",'3'
};
int mainMenuEntries = sizeof(mainMenu) / sizeof(menuEntryStruct);
```

功能执行部分完整代码如下：

```
void userEventAction(int key) {
    switch (key) {
    case '0':
        MyWorld.Empty();
        MyWorld.pickedObjectId = -1;
        MyState = MoveObject;
        break;
    case '1': // bezier
        if (MyState == MoveObject)
        {
            GLBezier * bezPtr = MyWorld.NewObject<GLBezier>();
            MyState = MakePoint;
            MyWorld.pickedObjectId = bezPtr->id;

        }
        else if (MyState == MakePoint)
        {
            MyState = MoveObject;
            MyWorld.pickedObjectId = -1;
        }
        break;

    case '2'://Auto Move
        if (MyState == MakePoint)
        {
            MyState = MoveObject;
            MyWorld.pickedObjectId = -1;
            break;
        }
        if (MyState == AutoMove)
        {
            MyState = MoveObject;
            break;
        }
        if (MyWorld.pickedObjectId == -1) {
            std::printf("Error:U've not choosen a point!\n");
            break;
        }
        std::printf("Auto Moving No.%d\n", MyWorld.pickedObjectId);
        MyState = AutoMove;
        glutTimerFunc(0, Auto, 0); // 启动定时器
        break;
    case 32://space
        if (MyState == AutoMove)
            MyState = MoveObject;
        break;
    case 'w':
        if (MyWorld.pickedObjectId != -1) {
            MyWorld.objects[MyWorld.pickedObjectId]->KeyboardMove(0, derta);
        }
        break;
    case 's':
        if (MyWorld.pickedObjectId != -1) {
            MyWorld.objects[MyWorld.pickedObjectId]->KeyboardMove(0, -derta);
        }
        break;
    case 'a':
        if (MyWorld.pickedObjectId != -1) {
            MyWorld.objects[MyWorld.pickedObjectId]->KeyboardMove(-derta, 0);
        }
        break;
    case 'd':
        if (MyWorld.pickedObjectId != -1) {
            MyWorld.objects[MyWorld.pickedObjectId]->KeyboardMove(derta, 0);
        }
        break;
    case 27: // ESC 键（ASCII: 27）退出
        MyWorld.Empty();
        exit(0);
        break;
    default:
        break;
    }
    glutPostRedisplay(); // 重绘
}
```

其中 wasd 用于键盘移动控制点，实现逻辑为调用对象的 KeyboardMove（）方法进行坐标运算

执行绘制贝塞尔曲线操作时，需要先判断当前状态是否为 AutoMove、MoveObject、MakePoint，若为 MoveObject，则开始绘制。若为 MakePoint 则结束绘制，释放 MyWorld.pickedObjectId 指针。若为 AutoMove，需要停止执行自动控制命令。

```
switch (state) {
    case GLUT_DOWN:
        if (MyState == MoveObject)
        {
            MyWorld.PickObject(x, y);
            std::printf("Pick No.%d\n", MyWorld.pickedObjectId);
            std::printf("U've Choosen:");
            for (auto ite = MyWorld.id.begin(); ite != MyWorld.id.end(); ite++) {
                std::printf("No.%d ", *ite);
            }
            std::printf("\n");
        }
        else if (MyState == MakePoint)
        {
            if(GLObject* ptr = MyWorld.objects[MyWorld.pickedObjectId])
            {
                // 从窗口到世界坐标系
                GLdouble winx = (GLdouble)x;
                GLdouble winy = (GLdouble)winHeight - (GLdouble)y;
                GLdouble winz = 0;
                GLdouble objx = 0;
                GLdouble objy = 0;
                GLdouble objz = 0;
                unProject(winx, winy, winz, &objx, &objy, &objz);
                ptr->MakePoint(objx, objy, objz);
            }
        }
        break;
    case GLUT_UP:
        if (MyState == MoveObject)
        {
            //MyWorld.pickedObjectId = -1;
        }
        glutPostRedisplay();
        break;
    }
```

利用 OpenGL 的 mouseMotion、mouseButton 自定义鼠标操作，用

于判断选中点、逆映射屏幕坐标生成控制点。鼠标按下时需要判断当前状态，若为MovePoint，需要判断是否选中控制点，若为MakePoint，需要调用 **unProject** 方法将屏幕坐标逆映射回世界坐标，生成控制点。

完整鼠标逻辑判断如下：

```
void mouseButton(int button, int state, int x, int y)
{
    switch (button) {
    case GLUT_LEFT_BUTTON:
        break;
    default:
        return;
    }
    switch (state) {
    case GLUT_DOWN:
        if (MyState == MoveObject)
        {
            MyWorld.PickObject(x, y);
            std::printf("Pick No.%d\n", MyWorld.pickedObjectId);
            std::printf("U've Choosen:");
            for (auto ite = MyWorld.id.begin(); ite != MyWorld.id.end(); ite++) {
                std::printf("No.%d ", *ite);
            }
            std::printf("\n");
        }
        else if (MyState == MakePoint)
        {
            if(GLObject* ptr = MyWorld.objects[MyWorld.pickedObjectId])
            {
                // 从窗口到世界坐标系
                GLdouble winx = (GLdouble)x;
                GLdouble winy = (GLdouble)winHeight - (GLdouble)y;
                GLdouble winz = 0;
                GLdouble objx = 0;
                GLdouble objy = 0;
                GLdouble objz = 0;
                unProject(winx, winy, winz, &objx, &objy, &objz);
                ptr->MakePoint(objx, objy, objz);
            }
        }
        break;
    case GLUT_UP:
        if (MyState == MoveObject)
        {
            //MyWorld.pickedObjectId = -1;
        }
        glutPostRedisplay();
        break;
    }
}

void mouseMotion(int x, int y)
{
    if (MyWorld.pickedObjectId == -1) return;
    if (!MyWorld.objects[MyWorld.pickedObjectId]->movable) return;

    GLObject * ptr = MyWorld.objects[MyWorld.pickedObjectId];

    // 从窗口到世界坐标系
    GLdouble winx = (GLdouble)x;
    GLdouble winy = (GLdouble)winHeight - (GLdouble)y;
    GLdouble winz = 0;
    GLdouble objx = 0;
    GLdouble objy = 0;
    GLdouble objz = 0;
    unProject(winx, winy, winz, &objx, &objy, &objz);

    ptr->SetPosition(objx, objy, objz);
    glutPostRedisplay();
}
```

综上，实现 Bezier 相关功能，测试如下：

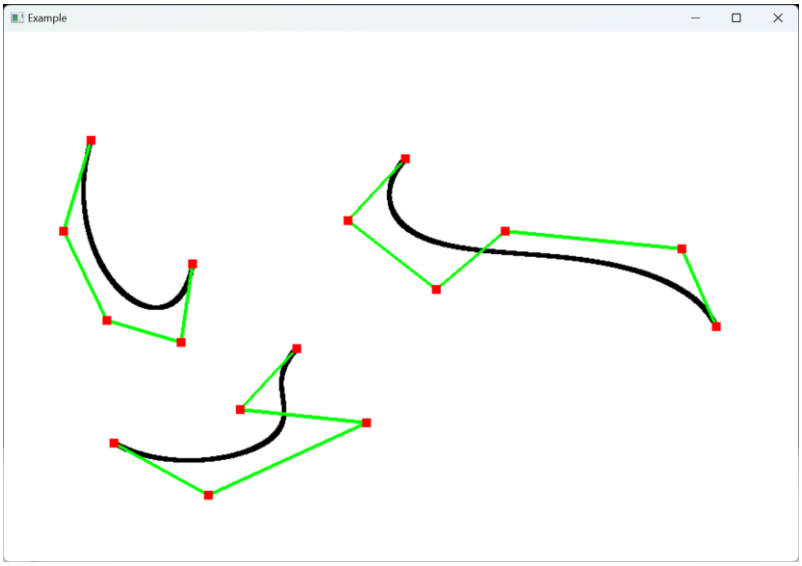


图 1-1 多 Bezier 曲线

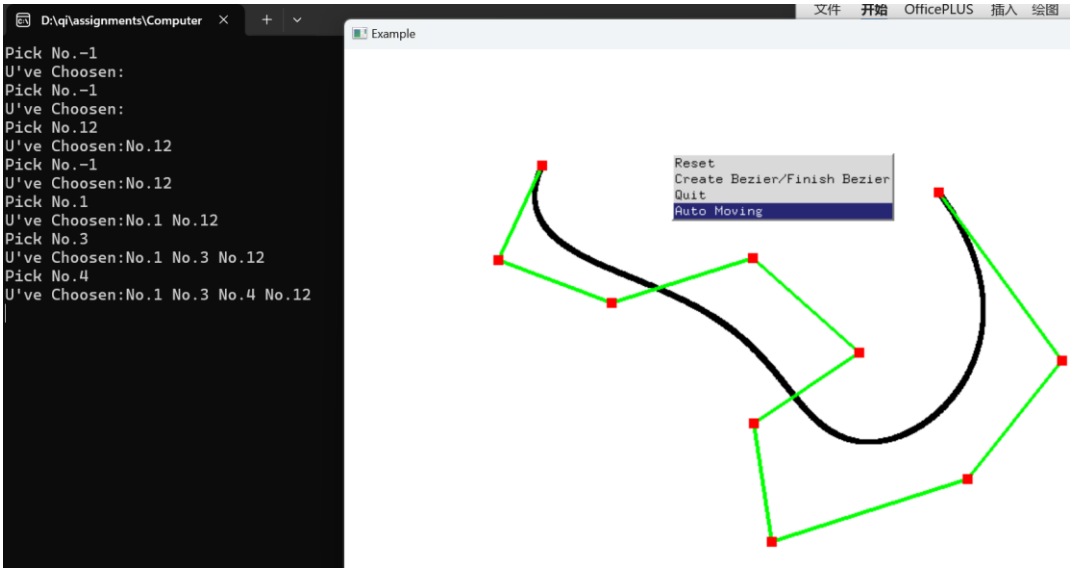


图 1-2 程序自动变换多控制点

2、分形几何

A)：定义视口坐标范围：设置四个视口，每个视口单独迭代

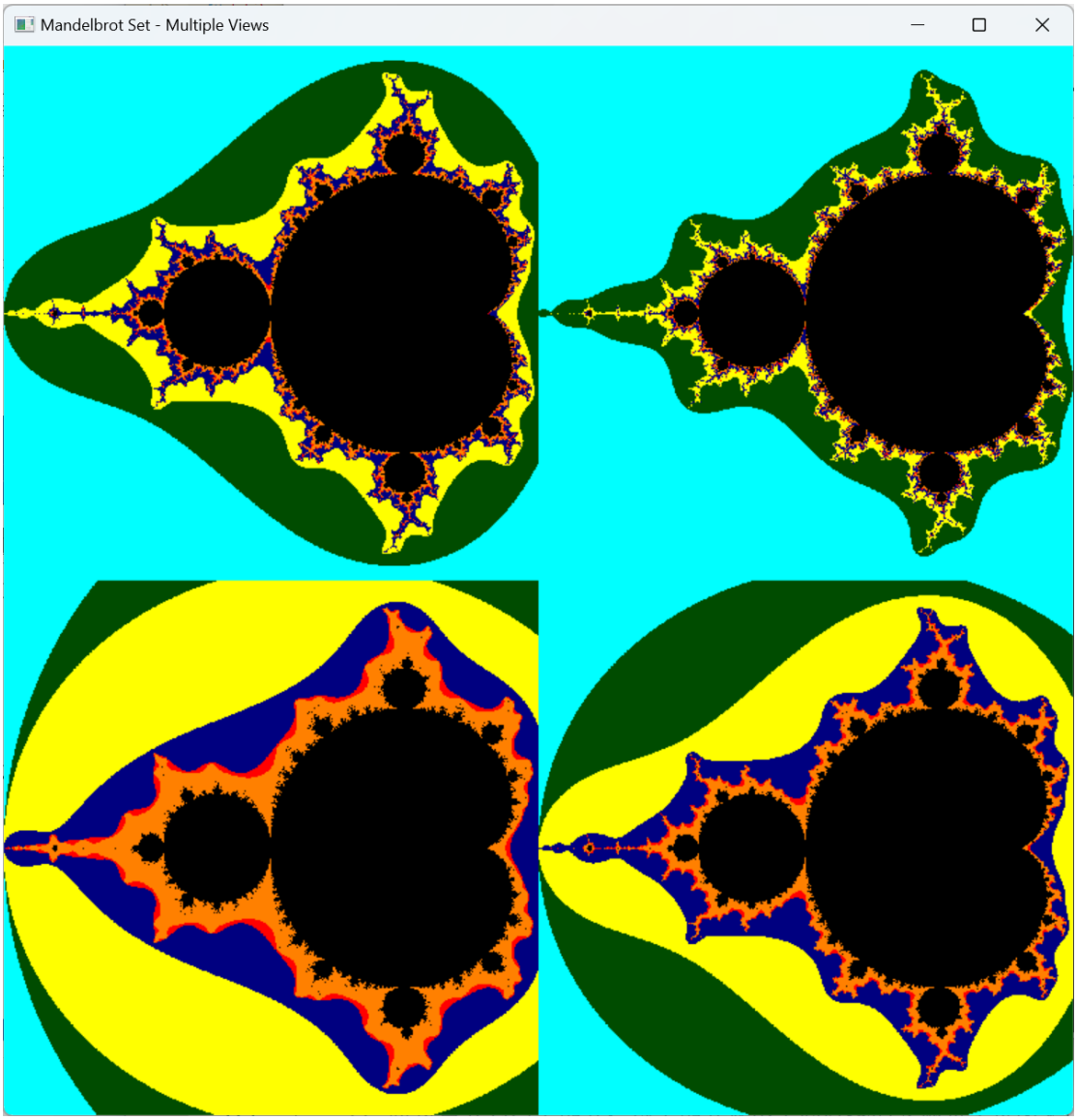
```
void displayFcn(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    // 定义四个区域的坐标范围
    GLfloat xMin[4] = { xComplexMin, xComplexMin + complexWidth / 2, xComplexMin, xComplexMin + complexWi
    GLfloat xMax[4] = { xComplexMin + complexWidth / 2, xComplexMax, xComplexMin + complexWidth / 2, xComp
    GLfloat yMin[4] = { yComplexMax - complexHeight / 2, yComplexMax - complexHeight / 2, yComplexMin, yC
    GLfloat yMax[4] = { yComplexMax, yComplexMax, yComplexMin + complexHeight / 2, yComplexMin + complexHe

    // 将窗口分为四个部分，每个部分显示不同区域的曼德布罗特图案
    for (int i = 0; i < 4; ++i) {
        // 设置视口
        glViewport((i % 2) * (winWidth / 2), (i / 2) * (winHeight / 2), winWidth / 2, winHeight / 2);  //

        // 绘制曼德布罗特集合，传递不同的坐标范围和最大迭代次数
        mandelbrot(nx, ny, maxIter[i]);
    }

    glFlush();
}
```

B）：自动迭代：

通过 glutTimerFunc(100, animate, 0); // 每 100 毫秒调用一次函数重复调用迭代方法，当达到迭代上限则停止

```
// 定时器回调函数，自动更新迭代次数
void animate(int value)
{
    for (int i = 0; i < 4; ++i) {
        if (maxIter[i] < maxIterLimit) {
            maxIter[i] += 10;  // 每次增加10次迭代
        }
    }
    glutPostRedisplay();  // 请求重绘
    glutTimerFunc(100, animate, 0);  // 每100毫秒调用一次
}
```

C）：测试如下：



五、总结及心得体会：

学习了 bezier 曲线构造、分形图形原理，体会到了 opengl 视角变换逻辑、世界坐标系和窗口坐标系的区别和转化

学习了分形几何相关构造原理，学会用多视口绘制的方法

六、对本实验过程及方法、手段的改进建议：

希望封装用户操作函数能在分形程序中指定迭代次数

报告评分：

指导教师签字：