吉林大学

软件学院

《图形学与人机交互》

实验报告

班级：

学号：

姓名：

2023-2024学年第1学期

|  |  |  |  |
| --- | --- | --- | --- |
| 实验项目 | 图形学算法实现与演示 | | |
| 实验性质 | 🗹演示性实验 □验证性实验  🗹操作性实验 □综合性实验 | | |
| 实验地点 | 计算机楼A108 | 机器编号 |  |
| 1. 系统实现的功能   实现了所有要求实现的功能，以下为各级菜单。   1. 图形绘制    1. 绘制矩形    2. 绘制圆形    3. 设置颜色 2. 区域填充    1. 绘制多边形    2. 设置颜色 3. 三维变换    1. 绘制立方体    2. 沿x轴方向平移    3. 沿y轴方向平移    4. 沿z轴方向平移    5. 绕x轴旋转    6. 绕y轴旋转    7. 绕z轴旋转    8. 设置数据 4. 绘制曲线    1. 绘制Bezier曲线   二、 实现的图形学算法  在本次项目中，为了成功实现功能，我大致上设计了两个类。  第一个类是图形基类（class Graph），主要用于实现各种平面图形的绘制，包括矩形、圆形，以及多边形。Graph类中的成员变量包括点集和颜色（此处颜色特指线条颜色）。Graph类并未实现具体的图形学算法，其主要功能是将平面图形类中的公共部分抽象出来，为子类提供公共的访问接口，主要是getter和setter方法。此外在Graph类中还定义了一个纯虚函数的void DrawGraph(CDC\* pDC)方法，由子类重写该方法，实现各自的图形学算法。  基于此类，分别派生出三个子类，分别矩形类（class RectangleGraph）、圆形类（class CircleGraph）、多边形类（class PolygonGraph），以及Bezier曲线类（class BezierGraph）。  就成员属性而言，RectangleGraph类、CircleGraph类和BezierGraph类，相较Graph类并无变化，主要是重写了void DrawGraph(CDC\* pDC)方法，以实现各自的图形学算法。  PolygonGraph类由于需要对图形进行填充，因此多了一个填充颜色的属性，此外便也是对void DrawGraph(CDC\* pDC)进行重写。  另一个类则是立方体类（class Cube），主要用于实现立方体的创建、投影、平移和旋转等功能。Cube类中包括了顶点位置、面顶点号、线面颜色、平移步长、旋转角度、透视投影视点等信息。  此外还有一些辅助性的类和方法，在此便不赘述。  以下具体介绍一下实现每个功能所依靠的函数和数据结构，其中涵盖了最为关键的图形学算法。不过其中一些与图形学算法并无太多关系的功能与函数，如图形绘制和区域填充中的设置颜色、三维变换中的设置数据等，我在此便不详细说明了。   1. **图形绘制**    1. **绘制矩形**       1. 实现函数   矩形的绘制主要依靠两个函数，分别是void DrawGraph(CDC\* pDC)和void DrawLine(CDC\* pDC, COLORREF color, int x1, int y1, int x2, int y2)。  void DrawGraph(CDC\* pDC)提供一个绘制矩形的接口。在此函数内得到矩形对角线上的两个顶点坐标，即鼠标点击和松开的坐标信息。然后依据这两个顶点的坐标信息，调用void DrawLine(CDC\* pDC, COLORREF color, int x1, int y1, int x2, int y2)，绘制矩形的四条边。  void DrawLine(CDC\* pDC, COLORREF color, int x1, int y1, int x2, int y2)的功能是连接(x1,y1)和(x2,y2)两个点，在两个点间绘制线段。这个函数中我使用的是DDA算法，即数值微分法。以下为两个函数的具体算法。   * + 1. 数据结构   实现矩形的绘制，除了矩形类本身，并没有使用什么复杂的数据结构。因为DDA算法本身只是一个对数学公式的简单模拟计算，并不需要多余的数据结构。   * 1. **绘制圆形**      1. 实现函数   圆形的绘制则主要依靠void DrawGraph(CDC\* pDC)。在这个函数中，我使用了中点画圆算法来实现圆形的绘制。通过鼠标按下和松开的信息，我们能够得到圆心和圆上一点的坐标信息，所以我们能通过这两个点的坐标信息和中点画圆算法绘制出圆形。以下为具体算法。   * + 1. 数据结构   与绘制矩形一样，圆形的绘制也无需复杂的数据结构。   1. **区域填充**    1. **绘制多边形**       1. 实现函数   绘制多边形主要依靠void DrawGraph(CDC\* pDC)和void EdgeMarkFill(CDC\* pDC, vector<CPoint> points)两个函数。  void DrawGraph(CDC\* pDC) 提供一个绘制的接口，包括了多边形绘制中的两个步骤，一个是点间连线，另一个则是区域填充。其中前者显然是较为简单的，可以直接复用绘制矩形时实现的void DrawLine(CDC\* pDC, COLORREF color, int x1, int y1, int x2, int y2)函数，以此完成点间连线的绘制。  至于第二步的区域填充，我将其单独提取为一个函数，即void EdgeMarkFill(CDC\* pDC, vector<CPoint> points)。这个函数中所使用的填充算法是边标志算法，即对每个像素访问一次。  边标志算法大体分为三个部分。首先是遍历顶点数组，得到多边形的边界值。然后是给多边形的边界打上标志，这也是这个算法中最难也最重要的一个部分。最后即是进行填充。   * + 1. 数据结构   边标志算法中也并未用到复杂的数据结构，最重要的即是MASK数组。MASK数组是一个bool类型的二维数组，用于给像素点搭上标志，后面便根据MASK数组中的数据进行颜色的填充。MASK数组是这个算法的核心之一，最后的填充质量很大程度上由其决定。   1. **三维变换**    1. **绘制立方体**       1. 实现函数   立方体的绘制主要依靠void DrawGraph(CDC\* pDC)。这个函数主要由三大部分组成，分别是投影、绘制边界和填充各面。  首先，在投影方面，这里使用的是透视投影。这一步最关键的是坐标在坐标系间的转换，也是这一功能中最核心的部分。为了得到三维立方体在平面上的透视投影坐标，我们要先转换到观察坐标系。在得到新坐标后，便可转换到屏幕坐标系，得到透视投影后各顶点的坐标。  在得到投影后的坐标后，便可进行边界，即各条边的绘制。显然，在这一步中我们能再次复用oid DrawLine(CDC\* pDC, COLORREF color, int x1, int y1, int x2, int y2)函数。  至于最后的填充步骤，如果不上色则能够免去这一步骤。   * + 1. 数据结构   在立方体的绘制过程中，最主要的是保存顶点坐标的两个数组，即Point viewPoints[8]和CPoint screenPoints[8]。此外，在具体的绘制过程中，哪些点共哪个面是必须要知道的，因此含有面顶点号的数组vector<vector<int>> mFacePoints便显得尤为重要。   * 1. **沿坐标轴方向平移**      1. 实现函数   实现沿坐标轴方向平移主要依靠void DrawGraph(CDC\* pDC)和void Translate(int axis, int direction)。  void Translate(int axis, int direction)用于得到立方体平移后的坐标位置。这个函数中接受两个参数，第一个参数axis用于指明移动的坐标轴，即立方体是沿哪一条坐标轴移动，第二个参数direction则用于指明移动的方向，即立方体是沿坐标轴的正方向移动还是沿坐标轴的负方向移动。  在得到立方体平移后的坐标位置后，只要再调用一次立方体的绘制函数void DrawGraph(CDC\* pDC)，便可实现平移效果。   * + 1. 数据结构   由于平移只是依靠坐标的加减，实现较为简单，所以没有用到什么复杂的数据结构。   * 1. **绕坐标轴旋转**      1. 实现函数   实现绕坐标轴旋转主要依靠void DrawGraph(CDC\* pDC)和void Rotate (int axis, int direction)。  void Rotate (int axis, int direction)用于得到立方体旋转后的坐标位置。这个函数中接受两个参数，第一个参数axis用于指明移动的坐标轴，即立方体是绕哪一条坐标轴转动，第二个参数direction则用于指明转动的方向，即是绕坐标轴正向转动还是绕坐标轴负向转动。  在得到立方体旋转后的坐标位置后，只要再调用一次立方体的绘制函数void DrawGraph(CDC\* pDC)，便可实现旋转效果。   * + 1. 数据结构   由于旋转只是依靠坐标的与三角函数相乘，实现较为简单，所以没有用到什么复杂的数据结构。   1. **绘制曲线**    1. **绘制Bezier曲线**       1. 实现函数   Bezier曲线的绘制主要依靠void DrawGraph(CDC\* pDC)、void SplitBezier(CDC\* pDC, vector<DoublePoint> points)、double MaxDistance(vector<DoublePoint> points)，以及void DrawPoint(CDC\* pDC)。  void DrawPoint(CDC\* pDC)用于标记出鼠标点击后的位置，使用一个5\*5的黑色实心矩形来标记Bezier曲线四个点的位置。因为知道矩形的中心位置和边长，只需要使用黑色实心画刷便可实现。  void DrawGraph(CDC\* pDC)提供一个绘制的接口，包括了绘制Bezier曲线的两个步骤。第一个步骤是将Bezier曲线的四个点连接起来，显然，只要复用void DrawLine(CDC\* pDC, COLORREF color, int x1, int y1, int x2, int y2)即可实现。第二个步骤则是绘制Bezier曲线本身，我将其单独封装为了一个函数，即void SplitBezier(CDC\* pDC, vector<DoublePoint> points)。  void SplitBezier(CDC\* pDC, vector<DoublePoint> points)用于计算并绘制Bezier曲线。我所使用的是分裂法，即通过多次分割曲线，使其逐渐逼近真实值。而算法的结束条件，即函数递归出口便是，当四个控制点之间的最大距离小于预先设定的epsilon。至于这最大距离的具体计算则由double MaxDistance(vector<DoublePoint> points)实现。  double MaxDistance(vector<DoublePoint> points)用于计算顶点距离底边的最大距离，通过计算两个矢量间的叉积和距离，得到两个矢量的投影长度的绝对值，并返回最大值。   * + 1. 数据结构   Bezier曲线的绘制，其核心在于分割二字，即通过大量的计算分割逐渐逼近真实值。值得注意的是在计算时要使用double类型的数据，因此不能直接使用自带的CPoint数据类型，否则会陷入无限递归，导致爆栈。  三、采用的交互方式   1. **图形绘制**    1. **绘制矩形和圆形**   绘制矩形和圆形的交互方式是都是通过鼠标进行交互。以鼠标按下的位置为起始坐标，以鼠标松开的位置作为结束坐标。对于矩形来说，鼠标按下的位置和鼠标放开的位置分别为对角线上的两个顶点。对于圆形来说，鼠标按下的位置为圆心坐标，鼠标送开的 位置为圆周上一点的坐标。  用到的系统信息包括void OnLButtonDown(UINT nFlags, CPoint point)、void OnLButtonUp(UINT nFlags, CPoint point)、void OnMouseMove(UINT nFlags, CPoint point)。   * 1. **设置颜色**   设置颜色主要是使用键盘输入，即输入图形线条的RGB值。  用到的系统信息包括void OnSetLineColor()（鼠标点击菜单后自动跳出对话框）。   1. **区域填充**    1. **绘制多边形**   绘制多边形是通过鼠标点击实现，鼠标点击的位置顺序即为多边形点边的顺序。若点击“Q”键位则可进行下一个的多边形的绘制。  用到的系统信息包括void OnLButtonDown(UINT nFlags, CPoint point)、void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags) 。   * 1. **设置颜色**   设置颜色主要是使用键盘输入，分别输入多边形线条颜色的RGB值和填充颜色的RGB值。  用到的系统信息包括void OnSetAreaColor()（鼠标点击菜单后自动跳出对话框）。   1. **三维变换**    1. **绘制立方体**   绘制立方体主要是通过键盘决定立方体的各面的颜色，即分别输入每个面的RGB值。  用到的系统信息包括void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags) 。   * 1. **沿坐标轴方向平移和绕坐标旋转**   沿坐标轴平移和绕坐标轴旋转都是靠键盘的输入来交互。在用鼠标选择好功能后，通过点击“A”和“L”键位，进行对立方体的操控，其中前者代表正方向，后者代表负方向。  用到的系统信息包括void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags) 。   * 1. **设置数据**   设置数据主要是使用键盘输入，分别输入立方体的平移步长和旋转角度。  用到的系统信息包括void OnSetData()（鼠标点击菜单后自动跳出对话框）。   1. **绘制曲线**    1. **绘制Bezier曲线**   绘制Bezier是通过鼠标实现交互。在点击屏幕四次后，便会自动绘制出Bezier曲线。  用到的系统信息包括void OnLButtonDown(UINT nFlags, CPoint point)。  四、实验结果   1. **图形绘制**    1. **绘制矩形**      * 1. **绘制圆形**      * 1. **设置颜色**      1. **区域填充**    1. **绘制多边形**      * 1. **设置颜色**      1. **三维变换**    1. **绘制立方体**      * 1. **沿x轴方向平移**      * 1. **沿y轴方向平移**      * 1. **沿z轴方向平移**      * 1. **绕x轴旋转**      * 1. **绕y轴旋转**      * 1. **绕z轴旋转**      * 1. **设置数据**      1. **绘制曲线**    1. **绘制Bezier曲线**     五、遇到的问题及解决办法  在本次大作业中，我遇到了许多问题，不过真正让我焦头烂额的并不多，对于一些小问题在此便不赘述。  我遇到的第一个大问题无关图形学算法，而是MFC框架的使用。由于是第一次使用MFC进行编程，代码该写在什么位置、对话框如何显示、控件如何绑定等，一开始都一无所知。因此，在刚开始的时候，由于对MFC的陌生，让我无从下手。之后通过教学视频、谷歌等方式，逐渐了解了MFC框架，初步学习了基于MFC实现编程。  我所遇到的第二个大问题是改变图像颜色。当时由于将绘图模式设置为R2\_NOT，导致无法进行颜色的修改。我开始一直以为是在代码传参或其他地方出了差错，耗费了大量时间，即使使用搜索引擎也无法搜索。我一直以为是算法方面的错误，但后来请教了同学才知道并不是。由于对绘图模式较为模式，导致在思考时忽略了这一关键要素。这件事也给我上了一课，让我明白思维惯性在思考时带来的劣势，以及在使用一个新开发工具前，应对其功能有个较为全面的认识。  以上两个问题与图形学算法本关系不大，但第三个问题确实是与图形学算法有关。在绘制立方体时，由于我立体几何数学知识较为薄弱，所以不知道如何进行透视投影，如何将三维空间中的坐标转换为透视投影的坐标。后来通过视频进行学习，这才成功实现了立方体的绘制。  最后一个问题则是Bezier曲线的绘制。绘制Bezier曲线的分裂法我是根据老师课件上的伪代码进行改写实现，但在运行时一直提示爆栈。在验证代码与老师课件确实一致后，我尝试去寻找别人在网上的开源代码，令人惊奇的是，我们的代码总体上几乎一致，也就是说我的算法因该是没有问题的。后来我将我的算法替换到此开源代码中，并略作修改，结果却是能够正常运行，对此我百思不得其解。  在思考了很久后我再次尝试分析爆栈的原因，爆栈我想多半是因为陷入了无限递归，即一直无法满足递归结束的条件，但我确信计算最大值的算法本身是没有问题的，那么问题很有可能来自数据。想到这我又再次将我的代码和开源的代码进行比较，我发现刚刚之所以能够运行成功，是因为我将控制点的数据类型由自带的CPoint转变为了开源代码中自定义的Point，而Point成员变量就是double x和double y，到这里我突然意识到可能是因为CPoint内部成员变量数据类型的问题，CPoint中使用的是int x和int y，然后我便自定义了DoublePoint结构，并将算法中的CPoint都用DoublePoint代替，问题果然得到了解决。  以上便是我在此次大作业中遇到几个的较大的问题，这几个问题耗费了我大量的时间。总体上而言，此次作业中最让我头疼的并不是图形学算法的实现，更多的是对框架的陌生，以及编程上一些不起眼但又至关重要的细节。 | | | |