**《计算机图形学》作业报告**

专业: **计算机科学与技术** 姓名: **陈嘉乐** 学号: **2021218152**

|  |
| --- |
| **1. 作业要求** |
| 本次作业使用上机实验平台和框架，实现多边形的扫描转换。具体任务包括：理解多边形扫描 转换的原理；掌握典型多边形扫描转换算法；掌握步处理、分析实验数据的能力；编程实现基本 X 扫描线转换算法（必做）；编程实现有效边表转换算法（选做）。 |
| **2. 作业内容 (重要步骤描述、关键代码段、结果截图、结果分析、作业心得)** |
| 1. **算法思路** 2. **X-扫描线法流程**    1. 确定顶点范围：遍历多边形的所有顶点，确定多边形在Y轴方向上的最小和最大值（min\_value\_y和max\_value\_y），作为扫描线的范围。    2. 扫描线处理：从最小Y值开始，逐行向上扫描至最大Y值。    3. 交点计算：对于每个扫描线，计算其与多边形各边的交点的X坐标。根据扫描线与多边形边的交点，将交点的X坐标存储起来。    4. 填充区域：对于每个扫描线，根据其与多边形边的交点，填充相应的像素。从每个扫描线的交点开始，将每对相邻的交点之间的像素填充为多边形内部。    5. 重复处理：按照扫描线的Y坐标顺序，重复处理每个扫描线，直到扫描完整个多边形的Y范围。 3. **有效边表转换算法流程**    1. 确定顶点范围：确定多边形顶点的 Y 坐标范围，找到最小和最大的 Y 坐标值。    2. 创建数据结构：创建活性边表头节点 pAET 和新边表数组 pNET，准备存储新边表的信息。    3. 构建新边表：对于每个 Y 坐标，遍历多边形的每个顶点。如果顶点的 Y 坐标与当前 Y 坐标相等，则检查相邻的顶点。如果相邻顶点的 Y 坐标大于当前顶点，则计算斜率并将该边加入新边表。    4. 构建活性边表：对于每个 Y 坐标，更新活性边表中各边的 X 坐标。对活性边表进行排序，按照 X 坐标从小到大的顺序排列边。删除活性边表中 ymax 等于当前 Y 坐标的边。将新边表中的边加入活性边表，并按照 X 坐标排序。使用活性边表填充扫描线之间的像素。    5. 结束：返回结果，完成多边形的填充。 4. **程序流程图**   确定多边形在Y轴方向上的最小和最大值  从每个交点出发，填充每对相邻交点之间的像素。  计算每个扫描线与多边形各边的交点的X坐标，并将交点的X坐标存储起来。  遍历多边形的各条边  确定多边形顶点的 Y 坐标范围  创建活性边表头节点 pAET 和新边表数组 pNET  如果顶点的 Y 坐标与当前 Y 坐标相等，则检查相邻的顶点。如果相邻顶点的 Y 坐标大于当前顶点，则计算斜率并将该边加入新边表。  对于每个 Y 坐标，更新活性边表中各边的 X 坐标。使用活性边表填充扫描线之间的像素。  结束   1. **关键代码段**   **（1）. X-扫描线法**  **void CPolygon\_ConversionView::X\_Scan\_Line\_Conersion(int Vertices[][2], int VertexNum)**  **{**  **int min\_value\_y = INT\_MAX, max\_value\_y = INT\_MIN;**  **int point\_x[10];**  **for (int i = 0; i < VertexNum; i++)**  **{**  **if (Vertices[i][1] < min\_value\_y)**  **{**  **min\_value\_y = Vertices[i][1];**  **}**  **if (Vertices[i][1] > max\_value\_y)**  **{**  **max\_value\_y = Vertices[i][1];**  **}**  **}**  **for (int ponit\_y = min\_value\_y; ponit\_y <= max\_value\_y; ponit\_y++)**  **{**  **int k = 0;**  **for (int m = 0; m < VertexNum; m++)**  **{**  **int j = (m + 1) % VertexNum;**  **if ((Vertices[m][1] <= ponit\_y && Vertices[j][1] > ponit\_y) || (Vertices[j][1] <= ponit\_y && Vertices[m][1] > ponit\_y))**  **{**  **point\_x[k++] = Vertices[m][0] + (ponit\_y - Vertices[m][1]) \* (Vertices[j][0] - Vertices[m][0]) / (Vertices[j][1] - Vertices[m][1]);**  **}**  **}**  **for (int n = 0; n < k - 1; n++)**  **{**  **for (int j = 0; j < k - n - 1; j++)**  **{**  **if (point\_x[j] > point\_x[j + 1])**  **{**  **int Intermediate = point\_x[j];**  **point\_x[j] = point\_x[j + 1];**  **point\_x[j + 1] = Intermediate;**  **}**  **}**  **}**  **for (int p = 0; p < k; p += 2)**  **{**  **for (int xi = min(point\_x[p], point\_x[p + 1]); xi <= max(point\_x[p], point\_x[p + 1]); xi++)**  **{**  **DrawPixel(xi, ponit\_y);**  **}**  **}**  **}**  **}**  **（2）. 有效边表转换算法**  **void CPolygon\_ConversionView::Active\_Edge\_Table\_Conersion(int PolygonVertices[][2], int VertexCount)**  **{**  **typedef struct edge\_table**  **{**  **float x; // 当前 x 坐标**  **float slope, ymax; // 斜率和最大 y 坐标**  **edge\_table \*next; // 指向下一个节点的指针**  **} AET, NET;**  **int maxY = PolygonVertices[0][1], minY = PolygonVertices[0][1];**  **int i;**  **for (i = 1; i < VertexCount; i++)**  **{**  **if (PolygonVertices[i][1] < minY)**  **minY = PolygonVertices[i][1];**  **if (PolygonVertices[i][1] > maxY)**  **maxY = PolygonVertices[i][1];**  **}**  **AET \*pAET = new AET;**  **pAET->next = NULL;**  **NET \*\*pNET = new NET \*[maxY - minY + 1];**  **for (i = 0; i <= maxY - minY; i++)**  **{**  **pNET[i] = new NET;**  **pNET[i]->next = NULL;**  **}**  **for (i = minY; i <= maxY; i++)**  **{**  **for (int j = 0; j < VertexCount; j++)**  **if (PolygonVertices[j][1] == i)**  **{**  **if (PolygonVertices[(j - 1 + VertexCount) % VertexCount][1] > PolygonVertices[j][1])**  **{**  **NET \*p = new NET;**  **p->x = PolygonVertices[j][0];**  **p->ymax = PolygonVertices[(j - 1 + VertexCount) % VertexCount][1];**  **p->slope = float(PolygonVertices[(j - 1 + VertexCount) % VertexCount][0] - PolygonVertices[j][0]) /**  **float(PolygonVertices[(j - 1 + VertexCount) % VertexCount][1] - PolygonVertices[j][1]);**  **p->next = pNET[i - minY]->next;**  **pNET[i - minY]->next = p;**  **}**  **if (PolygonVertices[(j + 1 + VertexCount) % VertexCount][1] > PolygonVertices[j][1])**  **{**  **NET \*p = new NET;**  **p->x = PolygonVertices[j][0];**  **p->ymax = PolygonVertices[(j + 1 + VertexCount) % VertexCount][1];**  **p->slope = float(PolygonVertices[(j + 1 + VertexCount) % VertexCount][0] - PolygonVertices[j][0]) /**  **float(PolygonVertices[(j + 1 + VertexCount) % VertexCount][1] - PolygonVertices[j][1]);**  **p->next = pNET[i - minY]->next;**  **pNET[i - minY]->next = p;**  **}**  **}**  **}**  **for (i = minY; i <= maxY; i++)**  **{**  **NET \*p = pAET->next;**  **while (p)**  **{**  **p->x = p->x + p->slope;**  **p = p->next;**  **}**  **AET \*tq = pAET;**  **p = pAET->next;**  **tq->next = NULL;**  **while (p)**  **{**  **while (tq->next && p->x >= tq->next->x)**  **tq = tq->next;**  **NET \*s = p->next;**  **p->next = tq->next;**  **tq->next = p;**  **p = s;**  **tq = pAET;**  **}**  **AET \*q = pAET;**  **p = q->next;**  **while (p)**  **{**  **if (p->ymax == i)**  **{**  **q->next = p->next;**  **delete p;**  **p = q->next;**  **}**  **else**  **{**  **q = q->next;**  **p = q->next;**  **}**  **}**  **p = pNET[i - minY]->next;**  **q = pAET;**  **while (p)**  **{**  **while (q->next && p->x >= q->next->x)**  **q = q->next;**  **NET \*s = p->next;**  **p->next = q->next;**  **q->next = p;**  **p = s;**  **q = pAET;**  **}**  **p = pAET->next;**  **while (p && p->next)**  **{**  **for (float j = p->x; j <= p->next->x; j++)**  **DrawPixel(static\_cast<int>(j), i);**  **p = p->next->next;**  **}**  **}**  **return;**  **}**   1. **结果截图**   左图为X边扫描算法，右图为多边形填充算法             1. **结果分析**   若从最终达成的效果来看，算法的实现可以说是相当成功的。它不仅能够顺利地填充凹凸多边形，而且也能处理更为复杂的图形填充工作，甚至在处理线段间的交点时也能展现出较高的效率与准确性。有效边表转换算法能够正确地填充多边形的内部，不留空洞并且不溢出多边形的边界。该算法在填充多边形时，仅考虑了与每条扫描线相交的边，而不是处理多边形内的每个像素点，因此具有较高的效率。  可以使用Y-扫描线算法来实现多边形的扫描转换和区域填充。与X-扫描线算法类似，Y-扫描线算法也是一种用于填充多边形的方法，其主要思想是沿着X坐标逐行扫描多边形，找到与多边形交点的Y坐标，然后填充扫描线之间的像素。  实现Y-扫描线算法：  1.确定Y坐标范围：遍历所有顶点，确定多边形的最小和最大Y坐标，作为扫描线的范围。  2.扫描线处理：对于每个扫描线，从最小Y坐标到最大Y坐标逐行扫描。  3.交点计算：对于每个扫描线，计算与多边形交点的X坐标。可以使用线段的斜率和截距来计算交点。  4.填充区域：根据计算得到的交点，填充扫描线之间的像素。  Y-扫描线算法与X-扫描线算法的主要异同点如下：  相同点：  1.都是用于填充多边形的算法。  2.都是基于扫描线的思想，逐行扫描多边形。  不同点：  1.X-扫描线算法是沿着Y坐标逐行扫描，而Y-扫描线算法是沿着X坐标逐行扫描。  2.在X-扫描线算法中，需要对每个扫描线找到与多边形相交的边，并计算交点的X坐标；而在Y-扫描线算法中，需要对每个扫描线找到与多边形相交的边，并计算交点的Y坐标。  3.填充像素的顺序也不同，X-扫描线算法是水平填充，而Y-扫描线算法是垂直填充。   1. **心得与思考**   通过本次实验我学习到了多边形填充算法和X边扫描算法，收获很多，是一次即扩展知识又进行实际动手操作的一次实验。在有效边表转换算法设计中，效率往往是一个重要的考量因素。这个算法通过只处理与扫描线相交的边，避免了处理多边形内的每个像素点，从而提高了效率。这让我意识到，在解决实际问题时，需要不断思考如何优化算法以提高效率。 |
| **提交日期： 2024 年 4 月 13 日** |