**《计算机图形学》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2022计算机科学与技术（卓越）01** | | | **姓名** |  |
| **实验题目** | 交互式B样条曲线设计 | | | | | |
| **实验时间** | **2025/4/7** | | **实验地点** | **DS3305** | | |
| **实验成绩** |  | | **实验性质** | **□验证性** √**设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  1. 掌握B样条曲线的基本概念；  2. 掌握de-Boor算法 | | | | | | |
| 二、实验项目内容  1．实现B样条曲线的交互设计程序和交互界面；  2．通过交互界面选择曲线阶数、顶点个数、节点个数和曲线上采样点的数量；  3．实现点的绘制和线段绘制两种方式；  4．实现通过鼠标改变控制点、对曲线进行几何变换； | | | | | | |
| 三、实验过程或算法（源程序）  本实验实现了一个具备完整交互功能的 B 样条曲线绘制系统，基于 OpenGL 和 ImGui 库，采用 C++ 编写，并可通过界面进行交互控制。  **1. B 样条曲线绘制算法实现**  B 样条曲线的关键在于基函数的递归计算和曲线点的组合。代码中使用以下两个函数实现该逻辑：  a. 递归基函数 BasisFunction()   1. float BasisFunction(int i, int k, float t, const std::vector<float>& knots)   B 样条曲线依赖一组称为“基函数”的权重函数来控制每个控制点对曲线形状的影响。程序中通过递归函数 BasisFunction 来计算这些权重。当阶数为 0 时（即最基础的情况），程序会判断当前参数值是否落在某个节点区间中，从而确定该控制点是否有效；当阶数大于 0 时，程序会递归调用两个较低阶的基函数，将它们组合起来计算出当前的权重值。​  b. 曲线点生成 CalculateBSpline()   1. std::vector<Point> CalculateBSpline()   一旦基函数定义完成，程序就可以开始生成曲线上的实际点。这个过程分为以下几步：首先根据当前控制点数量和阶数生成一个均匀的节点向量；然后，在参数范围内按等间距进行采样；对于每一个采样点，遍历所有控制点，计算其对当前曲线点的影响（即其基函数值），并将所有控制点的位置按权重叠加，得到一个曲线点；最后将所有生成的曲线点保存到一个列表中，供后续绘图使用。通过这一过程，即可生成一条光滑的 B 样条曲线，且可通过调整控制点位置、数量、阶数等参数动态变化曲线形状。  **2. 图形界面与参数控制**  界面使用 ImGui 实现，位于主循环中以下部分：   1. ImGui::SliderInt("Degree", &degree, 1, 5); 2. ImGui::SliderInt("Samples", &numSamples, 10, 200); 3. ImGui::Checkbox("Draw as Points", &drawAsPoints);   用户可以设置：1.曲线阶数 degree；2.曲线上采样点数 numSamples；3.控制点数量 controlPoints.resize(numPoints)；4.绘制模式（点或线段）drawAsPoints。  控制点的位置通过循环结构绘制坐标编辑器：   1. for (int i = 0; i < controlPoints.size(); ++i) { 2. ImGui::PushID(i); 3. ImGui::SliderFloat2("P", &controlPoints[i].x, 0.0f, 800.0f); 4. ImGui::PopID(); 5. }   **3. 控制点交互与几何变换**  控制点支持鼠标拖动，通过如下代码实现命中检测与拖动处理：   1. if (ImGui::IsMouseClicked(0)) { 2. for (int i = 0; i < controlPoints.size(); ++i) { 3. if (fabs(mousePos.x - controlPoints[i].x) < 8 && 4. fabs(mouseY - controlPoints[i].y) < 8) { 5. selectedPoint = i; 6. break; 7. } 8. } 9. }   拖动控制点时，根据鼠标偏移动态修改坐标：   1. if (selectedPoint != -1 && ImGui::IsMouseDragging(0)) { 2. ImVec2 delta = ImGui::GetMouseDragDelta(0); 3. controlPoints[selectedPoint].x += delta.x; 4. controlPoints[selectedPoint].y += delta.y; 5. ImGui::ResetMouseDragDelta(0); 6. }   另外，几何变换函数 ApplyTransformations() 将缩放、旋转和平移依次作用于所有采样点：   1. void ApplyTransformations(std::vector<Point>& points)   旋转角度通过 rotateAngle 控制，平移通过 translateX/Y 控制，缩放通过 scaleX/Y 控制。  **4. 曲线绘制模式：点 vs. 线段**  绘制部分根据变量 drawAsPoints 决定调用 GL\_POINTS 或 GL\_LINE\_STRIP：   1. if (drawAsPoints) { 2. glBegin(GL\_POINTS); 3. for (const auto& p : curvePoints) { 4. glVertex2f(p.x, p.y); 5. } 6. glEnd(); 7. } 8. else { 9. DrawCurve(curvePoints); 10. }   其中 DrawCurve() 函数定义如下：   1. void DrawCurve(const std::vector<Point>& points) { 2. glBegin(GL\_LINE\_STRIP); 3. for (const auto& p : points) { 4. glVertex2f(p.x, p.y); 5. } 6. glEnd(); 7. }   默认采用线段连接样条点的方式进行绘制，便于更直观地展现曲线的整体结构。  **5. OpenGL 渲染与坐标系设定**  在主循环中使用 glOrtho() 设置 2D 投影矩阵，确保以窗口左上角为原点的像素坐标系：   1. glOrtho(0, width, height, 0, -1, 1);   所有控制点与曲线点都使用窗口像素坐标渲染，保证用户交互直观。 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  **一、实验结果展示与功能分析**  1.B样条曲线绘制与交互界面设计  程序通过GLFW和ImGui创建了一个交互式窗口，用户可以实时调整B样条曲线的参数，并观察曲线形态的变化。ImGui提供的滑动条和输入框使得操作直观且响应迅速，提升了程序的可用性。    2.参数化控制：阶数、控制点、采样数量  在界面中，用户可调整：   * 曲线的阶数（Degree）范围为1至5； * 控制点数量通过“Control Points”输入框动态修改，自动补齐或截断； * 采样点数（Samples）从10到200，控制曲线精细程度； 这些参数直接影响了曲线的平滑程度与形状，可视化反馈良好。         3.绘制方式切换  提供“Draw as Points”复选框，允许用户选择以离散点形式显示曲线或以线段连接的方式绘制，适用于调试或细节观察。    4.控制点拖动与曲线实时响应  通过检测鼠标点击和拖动事件，程序支持控制点的选择与平移。当用户点击靠近控制点的位置并拖动时，曲线会实时更新，从而实现直观的交互式编辑。该功能有效验证了B样条曲线对控制点位置的局部影响特性。    5.几何变换功能  用户可通过输入框设置平移（Translate X/Y）、旋转（Rotate）和缩放（Scale X/Y）参数，对生成的曲线进行整体几何变换，变换后的结果立即在窗口中反映，验证了曲线图形的可变换性和灵活性。    **二、调试过程与问题分析**  1.控制点拖动的坐标系偏差  在鼠标交互中，GL坐标与ImGui获取的屏幕坐标存在y轴方向反向问题。通过使用glfwGetFramebufferSize获取窗口高度，并结合ImGui鼠标位置进行处理，使得拖动操作与实际显示坐标一致。  2.变换函数ApplyTransformations的作用顺序问题  初期对缩放、旋转、平移的顺序处理不当，导致最终变换结果不符合预期。经过调试，统一将变换顺序设定为“缩放 → 旋转 → 平移”，并使用标准的旋转公式以保证一致性。  3.实时刷新效率问题  为避免不必要的计算负担，采用分离式计算流程，即仅在参数或控制点改变时重新生成曲线点，提高了交互响应速度。 | | | | | | |