计算机学院实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目： 贝塞尔曲线 | | 学号：202300130183 |
| 日期：2025/3/24 | 班级：23级智能班 | 姓名：宋浩宇 |
| **Email：[2367651943@qq.com](mailto:2367651943@qq.com)**  **202300130183@mail.sdu.edu.cn** | | |
| 实验目的：  在确保代码框架一切正常后，就可以开始完成你自己的实现了。注释掉 main  函数中 while 循环内调用 naive\_bezier 函数的行，并取消对 bezier 函数的注  释。要求你的实现将 Bézier 曲线绘制为**绿色**。  如果要确保实现正确，请同时调用 naive\_bezier 和 bezier 函数，如果实  现正确，则两者均应写入大致相同的像素，因此该曲线将表现为**黄色**。如果是这  样，你可以确保实现正确。你也可以尝试修改代码并使用不同数量的控制点，来  查看不同的Bézier曲线。 | | |
| 实验环境介绍：  软件环境：  主系统：Windows 11 家庭中文版23H2 22631.4317  虚拟机软件：Oracle Virtual Box 7.1.6  虚拟机系统：Ubuntu 18.04.2 LTS  编辑器：Visual Studio Code  编译器：gcc 7.3.0  计算框架：OpenCV4  硬件环境：  CPU：13th Gen Intel(R) Core(TM) i9-13980HX 2.20 GHz  内存：32.0 GB (31.6 GB 可用)  磁盘驱动器：NVMe WD\_BLACKSN850X2000GB  显示适配器：NVIDIA GeForce RTX 4080 Laptop GPU | | |
| 解决问题的主要思路：  解决本问题的主要思路就是de Casteljau算法，该算法的运作方式是：对于在曲线上t（0-1）位置的点，它的位置由重复的的线性插值决定，具体方法为，求所有相邻控制点的线性插值点的位置，获得一组新的控制点，再对这些控制点进行线性插值，又获得一组新的点，重复这个步骤直到获得的点只有一个时，这个点就是贝塞尔曲线上的点。符号化的表达就是：  其中，最初的控制点为：  使用程序中的递归调用可以很简单地实现这个步骤，具体会在下个模块说明。 | | |
| 实验步骤与实验结果：  实验步骤：  我们在上文中说，这个算法用递归实现很简单，参考代码框架中给出的代码，我们每一层递归传递的信息是一个vector和此时的t值，我们在进行一轮线性插值时只需要将计算出来的点加入一个新的vector，并将这个vector作为参数传递到下一层递归，在vector中只有一个点时返回这个点即可。然后为了画出贝塞尔曲线，只需设置好步长，从0到1按照步长改变t的值并重复调用这个获得贝塞尔曲线的点的函数即可。  代码如下：   |  | | --- | | #include <chrono>  #include <iostream>  #include <opencv2/core/types.hpp>  #include <opencv2/opencv.hpp>  std::vector<cv::Point2f> control\_points;  size\_t control\_cnt = 0;  void mouse\_handler(int event, int x, int y, int flags, void\* userdata)  {    if (event == cv::EVENT\_LBUTTONDOWN && control\_points.size() < control\_cnt)    {      std::cout << "Left button of the mouse is clicked - position (" << x << ", "                << y << ")" << '\n';      control\_points.emplace\_back(x, y);    }  }  void naive\_bezier(const std::vector<cv::Point2f>& points, cv::Mat& window)  {    auto& p\_0 = points[0];    auto& p\_1 = points[1];    auto& p\_2 = points[2];    auto& p\_3 = points[3];    for (double t = 0.0; t <= 1.0; t += 0.001)    {      auto point = std::pow(1 - t, 3) \* p\_0 + 3 \* t \* std::pow(1 - t, 2) \* p\_1 +                   3 \* std::pow(t, 2) \* (1 - t) \* p\_2 + std::pow(t, 3) \* p\_3;      window.at<cv::Vec3b>(point.y, point.x)[2] = 255;    }  }  cv::Point2f recursive\_bezier(const std::vector<cv::Point2f>& control\_points,                               float t)  {    auto n = control\_points.size() - 1;    if (n == 0)    {      return control\_points[0];    }    std::vector<cv::Point2f> new\_points(n);    for (int i = 0; i < n; i++)    {      new\_points[i] = (1 - t) \* control\_points[i] + t \* control\_points[i + 1];    }    return recursive\_bezier(new\_points, t);  }  void bezier(const std::vector<cv::Point2f>& control\_points, cv::Mat& window)  {    // TODO: Iterate through all t = 0 to t = 1 with small steps, and call de    // Casteljau's recursive Bezier algorithm.    float step = 0.001;    float t = 0;    while (t <= 1)    {      cv::Point2f point = recursive\_bezier(control\_points, t);      window.at<cv::Vec3b>(point.y, point.x)[1] = 255;      t += step;    }  }  int main()  {    cv::Mat window = cv::Mat(700, 700, CV\_8UC3, cv::Scalar(0));    cv::cvtColor(window, window, cv::COLOR\_BGR2RGB);    cv::namedWindow("Bezier Curve", cv::WINDOW\_AUTOSIZE);    cv::setMouseCallback("Bezier Curve", mouse\_handler, nullptr);    std::cout << "请输入控制点个数：" << std::endl;    std::cin >> control\_cnt;    // control\_cnt = 4;    int key = -1;    while (key != 27)    {      for (auto& point : control\_points)      {        cv::circle(window, point, 3, {255, 255, 255}, 3);      }      if (control\_points.size() == control\_cnt)      {        // naive\_bezier(control\_points, window);        bezier(control\_points, window);        cv::imshow("Bezier Curve", window);        cv::imwrite("my\_bezier\_curve.png", window);        key = cv::waitKey(0);        return 0;      }      cv::imshow("Bezier Curve", window);      key = cv::waitKey(20);    }    return 0;  } |   实验结果：  首先是示例实验的贝塞尔曲线：  测试实验贝塞尔曲线  然后是我们自己实现的de Casteljau算法绘制出来的贝塞尔曲线。  因为我们通过设置全局变量获得了任意次的贝塞尔曲线的绘制函数，因此以下分别是4次的贝塞尔曲线和5次的贝塞尔曲线。  绿色四次贝塞尔曲线绿色五次贝塞尔曲线  最后是两种算法获得的贝塞尔曲线的重合状态：  两线重合 | | |
| 实验中存在的问题及解决：  问题1：OpenCV是怎么定义的颜色的值，为什么把点设置为红色修改的是颜色数组里的第三项的值？数组里的前两项是什么？  回答1：由于历史遗留原因，OpenCV中对于颜色的定义不是常见的RGB的顺序，而是BGR的顺序，因此想要将点设置为绿色，需要将数组中的第二项设置为255. | | |