Content

Introduction

The application generates the Bezier Curve by inputting control points. The user can also perturb the specific point to make the curve more stable.

Java is the primary language of this program. All *GUI* implementation uses the *Swing* and *AWT*(Abstract Window Toolkit). Swing is built on the top of *AWT* but extends some functionality.

Design and Implementation

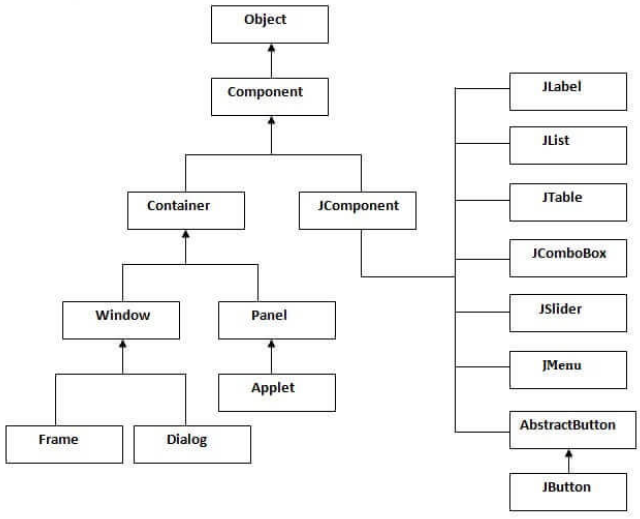

Figure 1: The hierarchy of java swing API

Windows Transition

The application contains windows:

- Menu for selecting the drawing pad
- Drawing pad to draw the control points and show the curves

The transition of this program is quite simple.

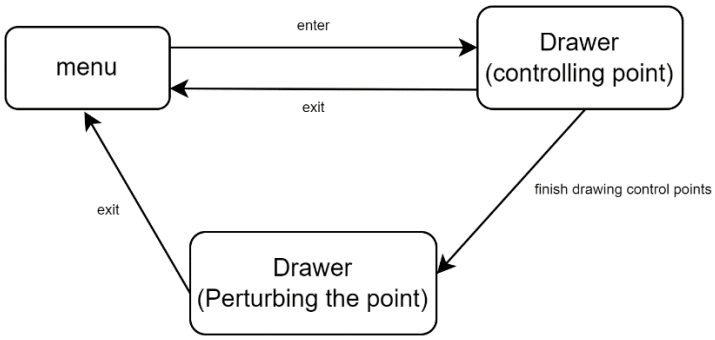

Figure 2:Transition of program

The *Container* class has the functionality of adding multiple *Component* objects. However, there are different relationships between the various *Component* objects. Therefore, the LayoutManager object is essential to identify its relationship with numerous component objects.

The *CardLayout* supports the ideology of the transition. It treats each *Component* object like a card by showing one at a time. Hence, the *Container* object shows another *Component* by triggering the *CardLayout* function.

The *Frame* and *Panel* classes inherit both *Component* and *Component* classes. The *Frame* class is the top-level window in the *Swing* application. Besides, the *Panel* class is the general-purpose container for the lightweight component. In general, a *Frame* object can not be the component of any other class, but the *Panel* object can.

The general structure of the program is a *Frame* object that has two components: the menu *Panel* and the drawer *Panel*. The *Frame* object uses the *CardLayout* to transit between different pages.

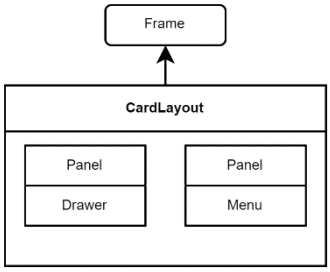

Figure 3:The Transition and CardLayout

User Interaction with Input

The user needs to select the control points to draw the bezier curve. The user first needs to determine the first and the last control point in the program. Then they can choose the random number of other control points.

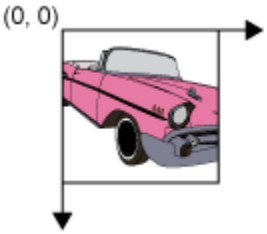

Figure 4:Coordinates in Java2D by Java Documentation

The method of selection is clicking the mouse on the screen. *MouseListener* class receives the user's input and record the coordinates of clicking. There is no hard code for identifying the specific coordinates, and *Java 2D* API does all the work.

Convex Hull and Graham's Scan

Once the user finishes clicking the control points, the program finds the convex hull for all the points by Graham's Scan.

The first step of Graham's scan is finding the leftmost bottom point of all points. The bottom point has the least y coordinate. If more than one point is at the bottom, the program will choose the leftmost point. Overall, the time complexity is O(n) in the worst case.

$$\cos\theta = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|}$$

Figure 5:Angle between two vectors

Secondly, the program connects the leftmost bottom point with the other point and derives the angle between the connecting line and the x-axis. The time complexity is O(n).

Thirdly, the program sorts those angles in order. I use the *TreeMap* class by setting the angle as key and the point as value. The time Complexity is O(*log n*) based on the *Java* documentation.

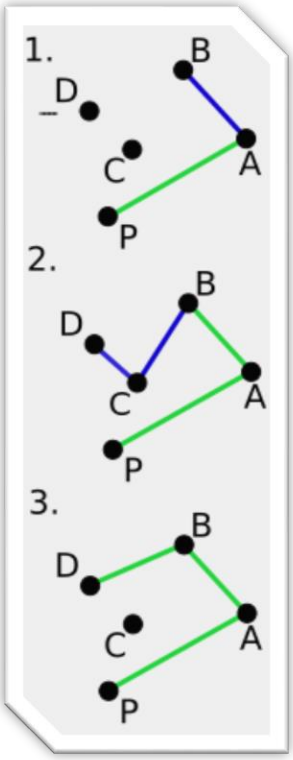

Figure 6:Step of Graham Scan from Wikipedia

After sorting the points, the program links the bottom point with the other two points in the angle's order. For example, in the picture above, the bottom point is P. Therefore, the connection series is P->A->B. The program checks if P->A->B is turning left. It is true in the example, so store P, A, B into the stack.

The checking sequence is P->A->B, A->B->C,B->C->D. In B->C->D, the turning direction is opposite visually, so delete the C from the stack and check B->D->P. The program keeps connecting until forming the convex hull. The time Complexity is O(n).

The equation to determine the turning direction is $(x_2 - x_1)(y_3 - y_1)- (y_2 -y_1)(x_3 -x_1)$ for $p_1(x_1 , y_1) , p_2(x_2, y_2) , p_3(x_3 , y_3)$. Negative value means turning right and vice versa.

Overall, finding the convex hull through Graham's scan led to time complexity O(*n log n*).

Bezier Curve Drawing

$$\mathbf{p}(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} \mathbf{p}_{i+1}$$

Figure 7:Equation derive the Bezier Curve

The calculation of the Bezier curve is quite intuitive. First, the program generates 100 sequential floating numbers between 0 and 1 for *u*. Hence, the program derives 100 points from the equation above to draw the Bezier curve. Finally, the program plots the Bezier curve by connecting those points straight. Increasing the sampling number can increase the smoothness of the curve but sacrifice the performance.

Bezier Curve Perturbation

The user can perturb the interior control points to achieve better stability in the curve.

Some references indicate that decreasing the degrees can increase the stability of the Bezier curve. Therefore, assume control points *a, b* and *c* are three adjacent points, not the starting or ending point. In this program, perturb the point *b* causes *b* to move closer to the line *bc* on the bisecting line of the angle between *ab* and *bc*. The perturbation stops when reaching the *bc*. It is the maximum perturbation level because the degree is reduced by one completely. The user achieves fix amount of perturbation each time they press that point until it ends.

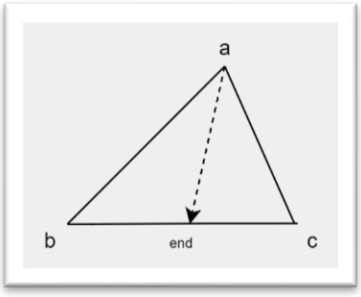

Figure 8:The perturbation

After indicating the user's input, the program will generate the unit vector for vector *ab* and *ac.* For example, let says it *ad* and *ae.* Then connect them to form an isosceles triangle because the unit vector has the same length. Hence find the middle point of the line *de* and connect that midpoint with point *a.* That line is the bisecting line of the angle between *ab* and *bc* based on the fundamental property of the isosceles triangle.

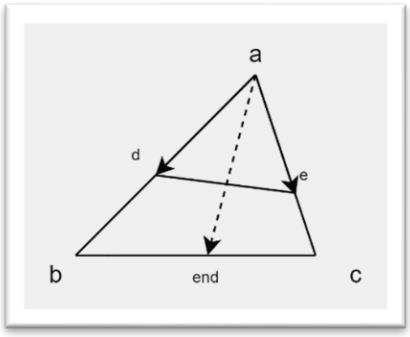

Figure 9: Finding the bisecting line

Let's name this bisecting line as line *af.* The program needs to know the length of the line *af* to make the perturbation adjustable. Firstly, the program derives the perpendicular distance from point *a* to line *bc* by line *ab* and its angle with line *bc.* Hence, the program concludes the length of line *af* from the angle between the perpendicular line and line *af* and the length of that vertical line.

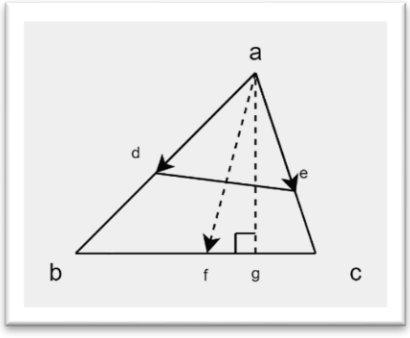

Figure 10: Derived the length of line *af*

3D Bezier Curve

The 3D Bezier curve uses the same function as the 2D version but uses one more coordinate from the z-axis. The difficult part of the 3D application is how to show it on the 2D screen. To show the 3D view, projection of the 3D graph to the 2D need to be implemented. The user can see the 3D view from the top right corner of the 3D diagram.

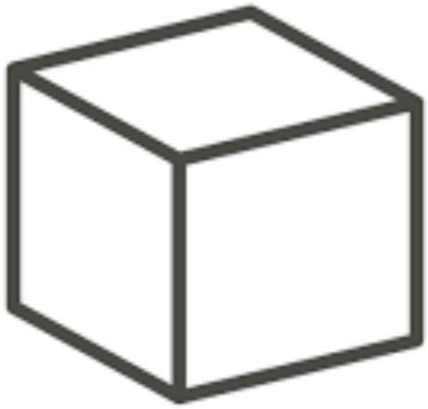

Figure 11: The best view to see the 3D view

The user can adjust the z-coordinate of the point in the possible extended version of the program. Then the programs derive the curve as usual by considering the existence of the Z-coordinate. In the final stage, the program shows the projection view of the 3D graph from the angle shown by the picture above.

Appendix

How to use the program

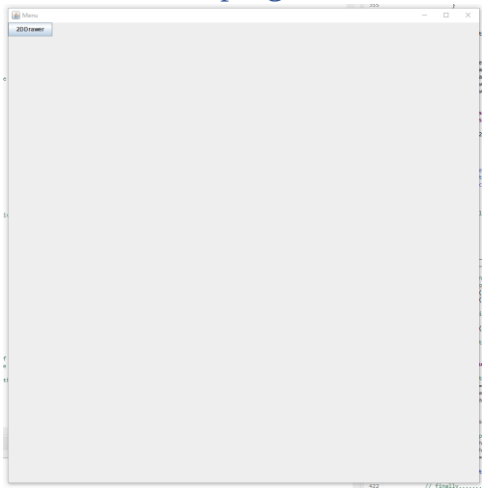

Figure 12: Menu, click the "2DDrawer"

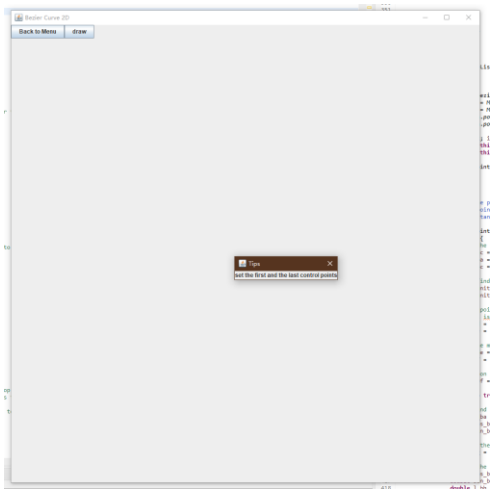

Figure 13: Close the Tips and draw the first and the last control points

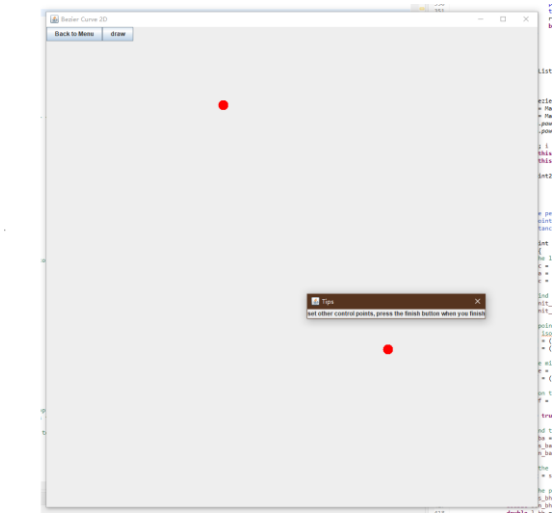

Figure 14:Close the tips and choose other control points

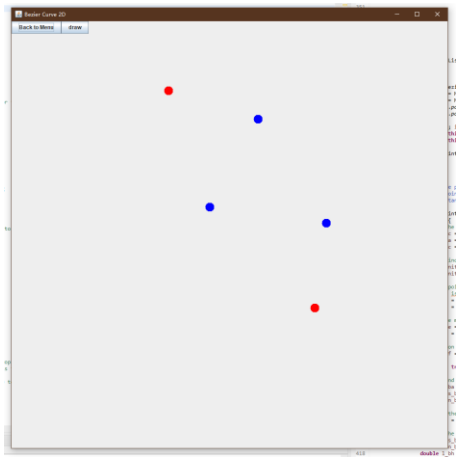

Figure 15: Red Point is the last and the first point, blue is the other point, once finish, press the "draw"

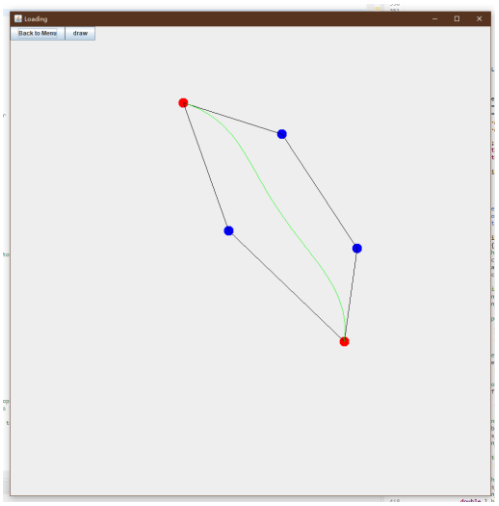

Figure 16: The Bezier curve and the Convex hull, press the blue point for perturbation

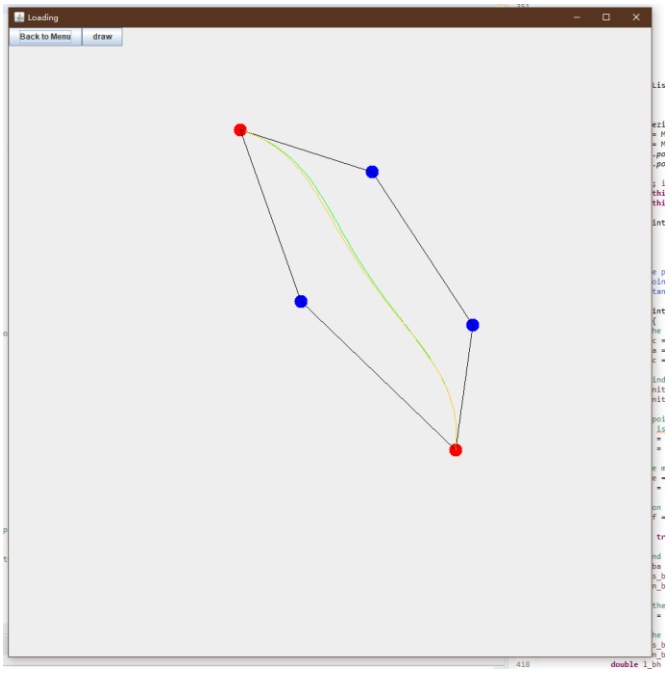

Figure 17 Orange line is the perturbed line. The green line is the original one, press more for more perturbation

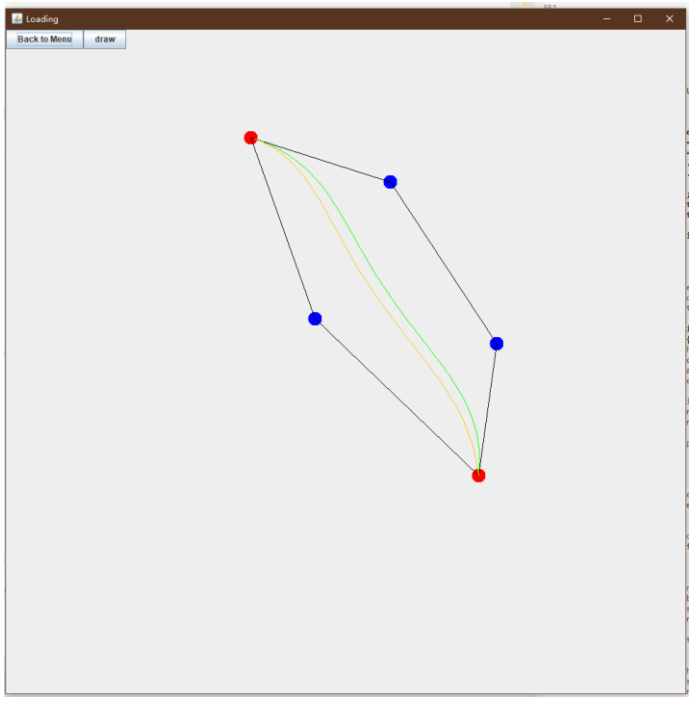

Figure 18: More Perturbation

More Example

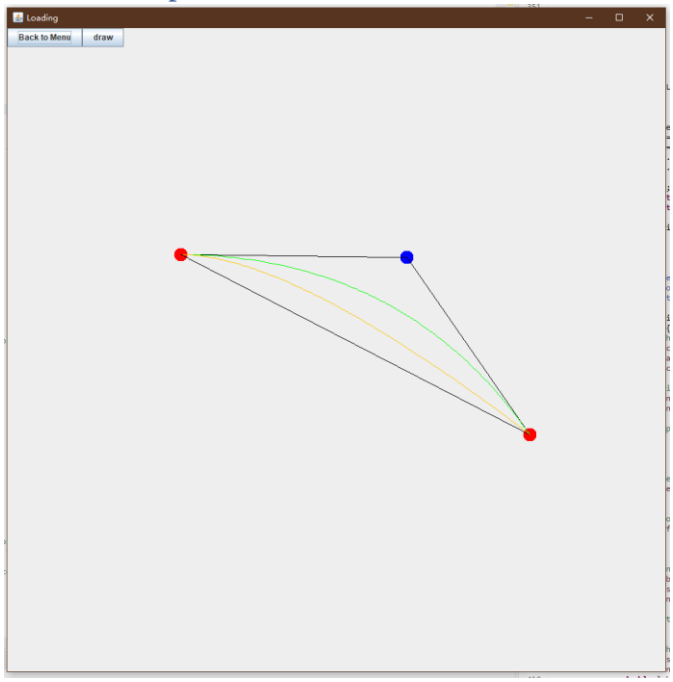

Figure 19:High level of perturbation