UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF

COMPUTING SEMESTER

1/20232024

SECJ3553 – GEOMETRIC MODELING

SECTION 01

ASSIGNMENT 2

Bezier/B-Spline Curve

LECTURER: DR. NORHAIDA BTE MOHD SUAIB

NAME	MATRIC NO
GAN HENG LAI	A21EC0176
NG KAI ZHENG	A21EC0101
LEW CHIN HONG	A21EC0044
YEO CHUN TECK	A21EC0148

Ways to control the shape of the generated curve(s)

The generated Bezier curve's shape is primarily controlled by the positions of the user-defined control points. In our program, the user can click on the canvas to add the control points. Therefore, the 'on_canvas_click' function was applied to take responsibility for adding control points and update the canvas accordingly.

```
def on_canvas_click(event):
    x, y = event.x, event.y
    control_points.append((x, y))

    coordinate_label.config(text=f"Mouse Click: ({x - 300}, {300 - y})")

    canvas.create_oval(x - 4, y - 4, x + 4, y + 4, fill="red")

    label_text = f"({x - 300}, {300 - y})"
    canvas.create_text(x + 20, y - 20, text=label_text, fill="blue", font=("Helvetica", 10))

    if len(control_points) >= 2:
        canvas.create_line(control_points[-2], control_points[-1], fill="blue")

    if len(control_points) >= 3:
        redraw_bezier_curve()
```

The main driver of our program is the application of De Casteljau's algorithm which will interpolate between those control points created by the user to create the curve. The algorithm determines the influence of each control point on the curve by computing intermediate points based on the interpolation parameter 't' value. Thus, the resulting curve becomes increasingly intricate and flexible as more control points are added. The shape of the curve can be influenced by the position of each control point.

```
def de_casteljau(control_points, t):
    if len(control_points) == 1:
        return control_points[0]
    else:
        new_points = []
        for i in range(len(control_points) - 1):
            x = (1 - t) * control_points[i][0] + t * control_points[i + 1][0]
            y = (1 - t) * control_points[i][1] + t * control_points[i + 1][1]
            new_points.append((x, y))
        return de_casteljau(new_points, t)
```

Additionally, the granularity of the interpolation is controlled by the 't_values' array in the 'redraw_bezier_curve' function. It will impact the overall smoothness of the curve. The Bezier curve is shaped by the interpolation parameter, control point placement and number, and overall interpolation.

```
def redraw_bezier_curve():
    canvas.delete("curve")
    t_values = [i / 100 for i in range(101)]
    curve_points = [de_casteljau(control_points, t) for t in t_values]
    for i in range(100):
        x1, y1 = curve_points[i]
        x2, y2 = curve_points[i + 1]
        canvas.create_line(x1, y1, x2, y2, fill="green", tags="curve")
```

Propose Suitable Controls and Interactions

To augment user control and facilitate curve modification, there already integrated several features and controls. Firstly, the mouse input enhancement allows users to drag existing control points to modify the shape of Bezier curve dynamically. It provides visual feedback when a point is selected or hovered over.

```
def on_canvas_motion(event):
    x, y = event.x, event.y

    coordinate_label.config(text=f"Mouse Position: ({x - 300}, {300 - y})")
```

The second interaction for the user to create Bezier curve is the undo functionality. The program implements the reset button functionality to allow users to revert changes made to the control points. Pressing the reset button will cause all the control points made by the users deleted from the screen. This feature enhances the user's ability to experiment with different curve shapes.

```
def reset_control_points():
    global control_points
    control_points = []
    canvas.delete("all")
    draw_coordinate_grid()
```

The third control and interaction is zoom and pan. Our program implements zoom and plan functionality to provide a better view of the curve. It can be proof when dealing with a large number of control points and approaching position between two control points.

```
def on_mousewheel(event):
    scale_factor = 1.1
    if event.delta > 0:
        canvas.scale("all", event.x, event.y, scale_factor, scale_factor)
    else:
        canvas.scale("all", event.x, event.y, 1/scale_factor, 1/scale_factor)
```

Ways to extend the 2D curve towards a 3D curved surface

Extending a 2D curve into a 3D surface involves fundamental steps and considerations. Initially, control points must be defined in three-dimensional space and each with coordinates (x, y, z). For a 3D curve to be adequately described, a suitable mathematical representation such as polynomials or parametric equations must be chosen. Using lofting algorithms becomes essential to join these three-dimensional control points smoothly and enable the production of a smooth, continuous surface. Customization is made possible by fine-tuning factors like tension and continuity, and realistic rendering and visualization techniques improve the 3D curved surface transformation's realistic portrayal. This procedure creates opportunities for a variety of applications from computer-aided design to computer graphics by using fundamental geometric principles.

References

1. “Computer Graphics: Principles and Practice” by James D. Foley, Andries van Dam, Steven K. Feiner, and John Hughes https://www.bing.com/search?q=Computer+Graphics%3A+Principles+and+Practice"+by+James+D.+Foley%2C+Andries+van+Dam%2C+Steven+K.+Feiner%2C+and+John+Hughes&cvid=11cee3288b9a424b89707b4fdeee4da2&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBBzI4OWowajGoAgCwAgA&FORM=ANAB01&PC=U531
2. “3D Surface plotting in Python using Matplotlib” from GeeksforGeeks https://www.geeksforgeeks.org/3d-surface-plotting-in-python-using-matplotlib/
3. “How do I convert a single 2D image to 3D using python?” https://www.quora.com/How-do-I-convert-a-single-2D-image-to-3D-using-python