Computer Graphics

11 - Lab - Curves

Yoonsang Lee
Hanyang University

Spring 2025

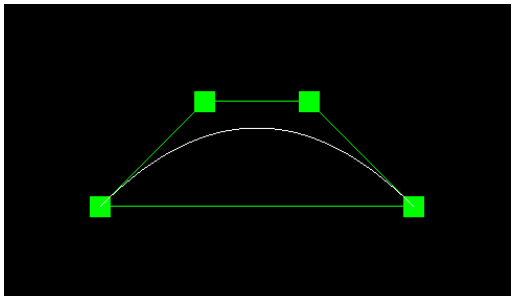
Outline

- Example: Interactive manipulation of a single Bezier curve

[Code] 1-interactive-cubic-bezier

- Drag a control point to manipulate the curve.

- Two VAOs
  - VAO for control points - `g_vao_control_points`
    - Green control points and control polygon
  - VAO for curve points - `g_vao_curve_points`
    - White curve



- One shader program
  - Control and curve points are rendered by the same shader program.
  - Their vertex attributes only have vertex positions.
  - Their colors are set through a uniform variable.

[Code] 1-interactive-cubic-bezier

- Global variable for the positions of control points:

```
g_control_points = [
    glm.vec3(250, 350, 0),
    glm.vec3(350, 450, 0),
    glm.vec3(450, 450, 0),
    glm.vec3(550, 350, 0),
    ]
```

- Global variable for the index of currently "dragging" control point:

```
g_moving_index = None
```

- They are defined as global variables to access and update them from main() and event callback functions.

[Code] 1-interactive-cubic-bezier

- Vertex shader

```
#version 330 core
layout (location = 0) in vec3 vin_pos;
uniform mat4 MVP;
void main()
{
    gl_Position = MVP * vec4(vin_pos, 1.0);
}
```

- Fragment shader

```
#version 330 core
out vec4 FragColor;
uniform vec3 color;
void main()
{
    FragColor = vec4(color, 1.0);
}
```

[Code] 1-interactive-cubic-bezier

```
def initialize_vao_for_points(points):
    # create and activate VAO (vertex array object)
    VAO = glGenVertexArrays(1)  # create a vertex array object ID and store it to
VAO variable
    glBindVertexArray(VAO)      # activate VAO

    # create and activate VBO (vertex buffer object)
    VBO = glGenBuffers(1)   # create a buffer object ID and store it to VBO variable
    glBindBuffer(GL_ARRAY_BUFFER, VBO)  # activate VBO as a vertex buffer object

    # only allocate VBO and not copy data by specifying the third argument to None
    vertices = glm.array(points)
    glBufferData(GL_ARRAY_BUFFER, vertices.nbytes, None, GL_DYNAMIC_DRAW)

    # configure vertex attributes
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * glm.sizeof(glm.float32),
None)
    glEnableVertexAttribArray(0)

    # return VBO along with VAO as it is needed when copying updated point position
to VBO
    return VAO, VBO
```

[Code] 1-interactive-cubic-bezier

```
def copy_points_data(points, vbo):
    glBindBuffer(GL_ARRAY_BUFFER, vbo)  # activate VBO

    # prepare vertex data (in main memory)
    vertices = glm.array(points)

    # only copy vertex data to VBO and not allocating it
    # glBufferSubData(target, offset, size, data)
    glBufferSubData(GL_ARRAY_BUFFER, 0, vertices.nbytes,
vertices.ptr)
```

[Code] 1-interactive-cubic-bezier

```
def generate_curve_points(control_points):
    curve_points = []

    for t in np.linspace(0, 1, 101): # linspace(start, stop, num)
        T = np.array([t**3, t**2, t, 1])

        # Bezier basis matrix
        M = np.array([[-1, 3, -3, 1],
                      [3, -6, 3, 0],
                      [-3, 3, 0, 0],
                      [1, 0, 0, 0]], float)

        P = np.array(control_points)
        p = T @ M @ P

        curve_points.append(glm.vec3(p))

    return curve_points
```

[Code] 1-interactive-cubic-bezier

```
def main():
    global g_vao_control_points, g_vao_curve_points
    global g_vbo_control_points, g_vbo_curve_points
    ...
    # prepare control points vao & vbo
    g_vao_control_points, g_vbo_control_points =
initialize_vao_for_points(g_control_points)
    copy_points_data(g_control_points, g_vbo_control_points)

    # generate initial curve points
    curve_points = generate_curve_points(g_control_points)

    # prepare curve points vao & vbo
    g_vao_curve_points, g_vbo_curve_points =
initialize_vao_for_points(curve_points)
    copy_points_data(curve_points, g_vbo_curve_points)

    # set point size (for drawing control points)
    glPointSize(20)

    while not glfwWindowShouldClose(window):
        ...
```

[Code] 1-interactive-cubic-bezier

```
    while not glfwWindowShouldClose(window):
        ...
        glUseProgram(shader_program)

        # projection matrix & set MVP uniform
        # to make camera space to have the same size as glfw screen space
        P = glm.ortho(0,WINDOW_WIDTH, 0,WINDOW_HEIGHT, -1,1)
        MVP = P
        glUniformMatrix4fv(unif_locs['MVP'], 1, GL_FALSE,
glm.value_ptr(MVP))

        # draw control polygon
        glUniform3f(unif_locs['color'], 0, 1, 0)
        glBindVertexArray(g_vao_control_points)
        glDrawArrays(GL_LINE_LOOP, 0, len(g_control_points))
        glDrawArrays(GL_POINTS, 0, len(g_control_points))

        # draw curve
        glUniform3f(unif_locs['color'], 1, 1, 1)
        glBindVertexArray(g_vao_curve_points)
        glDrawArrays(GL_LINE_STRIP, 0, len(curve_points))
        ...
```

[Code] 1-interactive-cubic-bezier

```
def hittest(x, y, control_point):
    if glm.abs(x-control_point.x)<10 and glm.abs(y-control_point.y)<10:
        return True
    else:
        return False

def button_callback(window, button, action, mod):
    global g_control_points, g_moving_index

    if button==GLFW_MOUSE_BUTTON_LEFT:
        x, y = glfwGetCursorPos(window)

        # convert from glfw screen coordinates (relative to the top-left corner)
        # to our camera space coordinates (relative to bottom-left corner)
        y = WINDOW_HEIGHT - y

        if action==GLFW_PRESS:
            g_moving_index = None
            for i in range(len(g_control_points)):
                if hittest(x, y, g_control_points[i]):
                    g_moving_index = i
                    break

        elif action==GLFW_RELEASE:
            g_moving_index = None
```

[Code] 1-interactive-cubic-bezier

```
def cursor_callback(window, xpos, ypos):
    global g_control_points, g_moving_index
    global g_vbo_control_points, g_vbo_curve_points

    # convert to our camera space coordinates
    ypos = WINDOW_HEIGHT - ypos

    if g_moving_index is not None:

        # update the moving control point position
        g_control_points[g_moving_index].x = xpos
        g_control_points[g_moving_index].y = ypos

        # copy updateded control point positions to g_vbo_control_points
        copy_points_data(g_control_points, g_vbo_control_points)

        # generated curve points from updated control points
        # and copy them to g_vbo_curve_points
        curve_points = generate_curve_points(g_control_points)
        copy_points_data(curve_points, g_vbo_curve_points)
```

Time for Assignment

- Let's start today's assignment.
- TA will guide you.