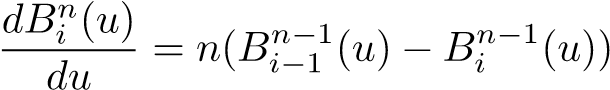
**COMP 5411 Advanced Computer Graphics** **Written Assignment**

# Problem 1. (20 points)

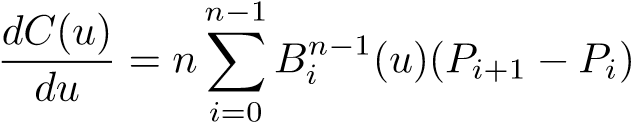
1. Show the following identity of the Bernstein polynomials

*,*

with

|  |
| --- |
| Answer :  A math equations with numbers  Description automatically generated with medium confidence |

1. Use the identity in (a) to derive the following general expression for the first derivative of an *n*-degree Bezier curve.



|  |
| --- |
| Recall that the Bézier curve defined by n + 1 control points P0, P1, ..., Pn has the following equation:  where is defined as follows:  Since the control points are constants and independent of the variable u, computing the derivative curve C'(u) reduces to the computation of the derivatives of Bn,i(u)'s.  Then, computing the derivative of the curve C(u) yields: |

1. Use the expression in (b) to obtain formulas for the end first derivatives of an *n*-degree Bezier curve.

|  |
| --- |
| The derivative of a Bézier curve is the difference of two Bézier curves of degree n-1. For simplicity, let these two curves be C1(u) and C2(u):  From these definitions, we know that the first curve C1(u) is defined by control points P1, P2, ..., Pn, that the second curve C2(u) is defined by control points P0, P1, ..., Pn-1, and that the derivative is: |

1. Differentiate the expression in (b) once more to derive a general expression for the second derivative of an *n*-degree Bezier curve (you may want to let *Qi* be *Pi*+1− *Pi*).

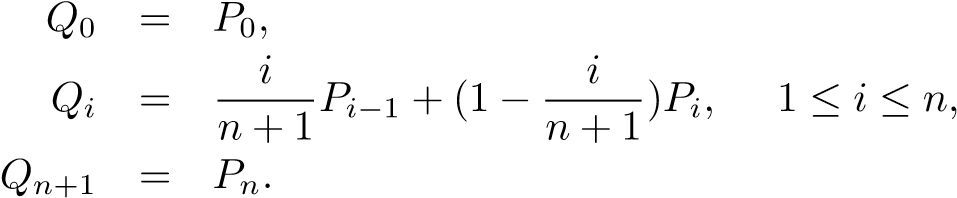
|  |
| --- |
| Recall that the derivative of C(u) is the following:  where  the second derivative of the Bézier curve can be calculated as following: |

1. Use the expression in (d) to obtain formulas for the end second derivatives of an *n*-degree Bezier curve.

To be asked for the teacher.

# Problem 2. (20 points)

Many applications require all involved curves to have the same degree. A polynomial of degree *n* is also a polynomial of degree *n*+1; therefore there exists a set of *n*+2 control points *Qi* that defines a degree *n* Bezier curve originally defined by *n* + 1 control points *Pi*. This process is called degree elevation. We give the relationship between *Qi* and *Pi* without proof here:



1. Sketch the control polygon of a cubic Bezier curve defined by (1*,*1)*,*(3*,*3)*,*(4*,*0)*,*(2*,*0). Sketch as accurate as possible the resulting control polygon after degree elevation.

|  |
| --- |
| (1) Here I used an online tool for visualizing the cubic Bezier curve defined by (1*,*1)*,*(3*,*3)*,*(4*,*0)*,*(2*,*0).  A screenshot of a computer  Description automatically generated    In our case, n = 3 (1*,*1)*,*(3*,*3)*,*(4*,*0)*,*(2*,*0).  So after degree elevation, it becomes as the figure below, we can see that it’s almost the same as the cubic Bézier curve.  A screenshot of a video game  Description automatically generated |

1. The corner-cutting procedure is similar to de Casteljiau algorithm. In what way are they different?

|  |
| --- |
| The corner-cutting procedure and de Casteljau's algorithm are both methods used to generate smooth curves or surfaces in computer graphics, but they differ in their approach.  The corner-cutting procedure involves iteratively removing corners from a polygon until it becomes a smooth curve or surface. This is done by connecting the midpoints of each edge and removing the original corners. This process is repeated until the polygon becomes a smooth curve or surface. The resulting curve or surface is a Bézier curve or surface. De Casteljau's algorithm, on the other hand, involves recursively dividing a Bézier curve or surface into smaller segments until the desired level of detail is achieved. This is done by computing intermediate control points along the curve or surface. These intermediate control points are then used to generate smaller Bézier curves or surfaces, which are combined to form the final curve or surface. |

# Problem 3. (20 points)

1. Consider a degree *n* Bézier curve . How many linear interpolations are involved to evaluate *C*(*u*) at a fixed value of *u* using the de Casteljau algorithm?

|  |
| --- |
| The algorithm works by recursively dividing the curve into two segments and computing intermediate control points. At each level of recursion, we need to compute n-1 new control points by interpolating between adjacent control points. This gives us a new set of n control points, which we use to recursively subdivide the curve again until we reach a single point, which is the value of the curve at the fixed value of u. |

1. Given a (*m* × *n*)-degree Bézier surface *S*(*u,v*) defined as

**

How many linear interpolations are required to evaluate the surface at fixed parameter values (ˆ*u*, ˆ*v*) using the de Casteljau algorithm? Note that there are two different answers.

|  |
| --- |
| Depending on the order in which the de Casteljau algorithm is applied to the parameters u and v, there are two different answers.  If we apply the de Casteljau algorithm to u first and then to v, we need to perform m\*(n-1) interpolations for each value of u, and then n-1 interpolations to compute the final value of S(ˆu, ˆv) using the intermediate values obtained from the first step. Therefore, the total number of interpolations required is m\*(n-1) + n-1 = (m+1)\*(n-1).  If we apply the de Casteljau algorithm to v first and then to u, we need to perform n\*(m-1) interpolations for each value of v, and then m-1 interpolations to compute the final value of S(ˆu, ˆv) using the intermediate values obtained from the first step. Therefore, the total number of interpolations required is n\*(m-1) + m-1 = (m-1)\*(n+1).  Input: a m+1 rows and n+1 columns of control points and (u,v).  Output: point on surface p(u,v)  Algorithm:  for i := 0 to m do  begin  Apply de Casteljau's algorithm to the i-th row of control points with v;  Let the point obtained be qi(v);  end  Apply de Casteljau's algorithm to q0(v), q1(v), ..., qm(v) with u;  The point obtained is p(u,v); |

1. Assuming *m < n*, how would you perform the evaluation in part (b) so as to minimize the number of linear interpolations needed?

|  |
| --- |
| Count\_case\_1=(m+1)\*(n-1)  Count\_case\_2=(m-1)\*(n+1)  Count\_case\_1 - Count\_case\_2 = (m+1)\*(n-1)  - (m-1)\*(n+1) = 2(n-m)   1. So if n <= m, then we apply the de Casteljau algorithm to u first and then to v. Otherwise, we apply the de Casteljau algorithm to v first and then to u, ) so as to minimize the number of linear interpolations needed. |

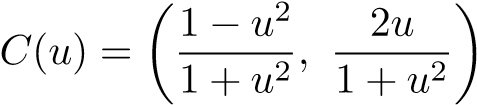
# Problem 4. (20 points)

Given a cubic Bézier curve defined by the control points *Pi*, *i* = 0*,*1*,*2*,*3. Explain how you would find the four B-spline control points of the same polynomial curve.

|  |
| --- |
| For a cubic B-spline curve, we can use a uniform knot vector which is defined as below:    Then we get the knot vector as: u0 = u1 = u2 = 0, u3 = u4 = u5 = 1  Then it’s able to the B-spline control points with the following formula:  where Pi-1, Pi, and Pi+1 are the Bézier control points adjacent to Bi.  Therefore, the four B-spline control points of the cubic Bézier curve are: |

# Problem 5. (20 points)

The first quadrant of the unit circle can be represented by the following quadratic rational functions:

*,* 0 ≤ *u* ≤ 1*.*

Let us derive the quadratic rational Bezier representation of this circular arcs from the above functions.

1. Based on the endpoint interpolation property and the first derivative of rational Bezier curves at *u* = 0 and *u* = 1, find *P*0*, P*1 and *P*2.

|  |
| --- |
| *At u* = 0 and *u* = 1, we can obtain 2 starting points: *P*0 = (1, 0), and *P*2= (0, 1).  P0 = (1, 0), P1 = (1, 1), and P2 = (0, 1). |

1. Noting that. Derive the weights *w*0*, w*1*, w*2.

|  |
| --- |
| Comparing the coefficients, we get:  So : |

1. Write in homogeneous space.

|  |
| --- |
|  |

1. Write the non-rational (polynomial) Bezier curve *Cw*(*u*) in terms of. Explicitly write out the Bezier basis functions and the coordinates of.

|  |
| --- |
| Where the Bezier basis functions.  Notice that when = ··· = , non-rational, or integral, Bézier curve is obtained.  Let = (, , ) and defined the homogeneous control points by |

1. Write the coordinate functions *X*(*u*) and *Y* (*u*) of the rational Bezier curve *C*(*u*) by performing a perspective mapping. Show that they indeed equal the rational functions given above.

|  |
| --- |
| To show that the coordinate functions X(u) and Y(u) obtained by performing a perspective mapping are equal to the given functions x(u) and y(u), we need to apply the perspective transformation to the control points and compute the new coordinate functions. Let's assume a perspective transformation with d = 1.  The perspective transformation matrix M is:  M = [1 0 0 0] [0 1 0 0] [0 0 1 1] [0 0 -1 0]  Transforming the control points P0, P1, and P2 using the matrix M, we get:  P0' = (1, 0, 0, 1) P1' = (1 + √2/√5, √2/√5, 0, 1) P2' = (0, 1, 0, 1)  Dividing the transformed control points by their homogeneous coordinates, we get:  X(u) = [(1 - u^2) + (1 + √2/√5) \* 2u(1 - u) + 0] / [(1 - u)^2 + 2(1 + √2/√5)u(1 - u) + u^2] Y(u) = [0 + √2/√5 \* 2u(1 - u) + 1] / [(1 - u)^2 + 2(1 + √2/√5)u(1 - u) + u^2]  Simplifying these expressions, we get:  X(u) = (1 - u^2) / (1 + u^2) Y(u) = 2u / (1 + u^2)  These are the same as the given functions x(u) and y(u), which confirms that the perspective mapping has been correctly applied to the Bezier curve. |