Assignment 2 : Geometric Modeling

NAME: BINGNAN LI
STUDENT NUMBER: 2020533092
EMAIL: LIBN@SHANGHAITECH.EDU.CN

1 INTRODUCTION

- Implemented evaluation function of Bézier curve with De Casteljau's algorithm.
- Successfully constructed and drew Bézier surface (take Tea Party as an example)
- Implemented adaptive mesh.
- Implemented B-Spline mesh.

2 IMPLEMENTATION DETAILS

(1) **Bézier Curve:**

With the recurrence formula

$$P_{n,m} = (1-u)P_{n-1,m-1} + uP_{n-1,m}$$

we can calculate the position of Bézier curve point with parameter u. Moreover, given that the line between $P_{n-1,m-1}$ and $P_{n-1,m}$ is tangent to Bézier where $P_{n,m}$ is in curve, we can calculate the normal vector of point $P_{n,m}$ by calculating $P_{n-1,m-1} - P_{n-1,m}$.

Pseudocode:

Algorithm 1 evaluate Bézier Curve

input: array $P[0:n]$ and real number u in $[0,1]$
output: position and normal on curve
for $i := 0 \rightarrow n$ **do**
  Q[i]=P[i]
end for
for $k := 1 \rightarrow n$ **do**
  for $i := 0 \rightarrow n-k$ **do**
    if $i == n-k$ **then**
      $normal := Q[i] - Q[i+1]$
    end if
    $Q[i] = (1-u)Q[i] + uQ[i+1]$
  end for
end for
return Q[0], normal

(2) **Bézier surface:**

(a) **Evaluation:**

We can get the position of point with parameters u, v by evaluating points along u direction with parameter u and utilize those new points as new control points and evaluate the final point at parameter v.

As for normal vector, we can perform the procedure above twice but in different orders. Namely, in the first turn, we evaluate the points with order u, v and get corresponding normal vector along v direction. Then we perform the procedure in v, u direction and get the corresponding normal vector. Lastly, we calculate the cross product of those two normal vector in order to get the normal vector of Bézier surface.

Pseudocode:

Algorithm 2 evaluate Bézier surface

input: 2-d array $P[0:n][0:m]$ and real numbers u, v in $[0,1]^2$
output: position and normal on surface
for $i := 0 \rightarrow n$ **do**
  for $j := 0 \rightarrow m$ **do**
    M[i][j] = P[i][j]
    N[j][i] = P[i][j]
  end for
end for
for $i := 0 \rightarrow n$ **do**
  line add $evaluate(M[i], u)$
end for
$position, normal_1 = evaluate(line, v)$
for $i := 0 \rightarrow m$ **do**
  line add $evaluate(N[i], v)$
end for
$position, normal_2 = evaluate(line, u)$
$normal = cross(normal_1, normal_2)$
return position, normal

(b) **Triangulation:**

Firstly, I divided $[0,1]$ into n and m pieces uniformly and treat those two sequences as sampling points parameters of Bézier surface. After that, we actually get the sampling grid.

Then, we iterate through the sampling grid, for each point (u_i, v_j), we construct two triangles:

$$triangle1: \ (u_i, v_j), (u_{i+1}, v_j), (u_{i+1}, v_{j+1})$$
$$triangle2: \ (u_i, v_j), (u_i, v_{j+1}), (u_{i+1}, v_{j+1})$$

Pseudocode:

Algorithm 3 Triangulation

input: sample number I, J > 0
output: indices for EBO
for $i := 0 \rightarrow I - 1$ **do**
  for $j := 0 \rightarrow J - 1$ **do**
    $index.add(i * (J + 1) + j)$
    $index.add((i + 1) * (J + 1) + j)$
    $index.add((i + 1) * (J + 1) + (j + 1))$
    $index.add(i * (J + 1) + j)$
    $index.add(i * (J + 1) + (j + 1))$
    $index.add(i * (J + 1) + j)$
  end for
end for
return index

(c) **Multi-Bézier-surface**

Since the given models are already processed to have L1 continuity, I just need to read in model from .bzs file, evaluate each surfaces via corresponding control points and draw them one by one.

(3) **B-Spline surface:**

(a) **Knot Vector:**

For n given control points and degree p, the number of knots m is

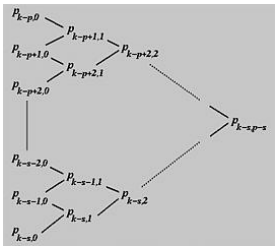
$$m = n + p + 1$$

In order to get a clamped B-spline curve, the multicity of the first and last knot must be $p + 1$. Then for the rest $m - 2(p + 1)$ knots, I uniformly set them between $(0, 1)$.

(b) **De Boor's Algorithm:**

By De Boors' Algorithm, if we repeatedly insert knots u until its multicity grows to p, the last generated control point is just on the curve.

Then for each input u, we need to determine which control points are used to generate new control point.



Based on the picture above, we know for input $u \in [knot_k, knot_{k+1})$, we need control points P_{k-p} to P_{k-s} to generate new control point.

By De Boor's Algorithm, we have a recurrence formula

$$P_{n,m} = (1 - a_{n,m})P_{n-1,m-1} + a_{n,m}P_{n,m-1}$$

where

$$a_{n,m} = \frac{u - knot_n}{knot_{n+p-r+1} - knot_n}$$

Pseudocode:

Algorithm 4 B-Spline

input: a value u
output: the position and normal of point on curve
if $u \in [u_k, u_{k+1})$ and $u \neq u_k$ **then**
  h=p
  s=0
end if
if $u = u_k$ and u_k has multiplicity s **then**
  h=p-s
end if
for $r := 1 \rightarrow h$ **do**
  for $i := k - p + r \rightarrow k - s$ **do**
    if $i == k - s$ **then**
      $normal = p_{i-1,r-1} - p_{i,r-1}$
    end if
    $a_{i,r} = (u - u_i)/(u_{i+p-r+1} - u_i)$
    $P_{i,r} = (1 - a_{i,r})P_{i-1,r-1} + a_{i,r}P_{i,r-1}$
  end for
end for
return $P_{k-s,p-s}$ and $normal$

(4) **Adaptive Mesh:**

I use curvature as an indicator of termination. However, curvature is hard to compute, so I use an alternative way to approximate curvature: for any control points $\{P_i\}_{i=1}^4$, I compute the sum of distances from P_2 to line P_1P_4 and P_3 to line P_1P_4 as a substitution of curvature. Once the sum of distances less than $factor \times length(P_1, P_4)$, we add the point with parameter 0.5 into adaptive sample points. Else I recursively compute on the left side and right side.

However, the only problem is that if I want to use this algorithm to generate surface, the four lines evaluated along u direction are not guaranteed to contain the same number of points, which means we can not perform this algorithm in v direction.

In order to solve this problem, I generated several intervals of u and v and perform adaptive curve algorithm in two directions. For interval $[u_t, u_{t+1})$ or $[v_t, v_{t+1})$, I count how many adaptive points lay on this interval and take the largest density and uniformly divide this interval to construct new sample grid.

After we get new sample grid, I evaluate the new surface via the new sample grid.

Pseudocode:

3 RESULTS

- Singal-Bézier-surface

Algorithm 5 B-Spline

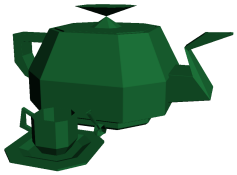
input: a value u
output: the position and normal of point on curve
if $u \in [u_k, u_{k+1})$ and $u \neq u_k$ **then**
  h=p
  s=0
end if
if $u = u_k$ and u_k has multiplicity s **then**
  h=p-s
end if
for $r := 1 \rightarrow h$ **do**
  for $i := k - p + r \rightarrow k - s$ **do**
    if $i == k - s$ **then**
      $normal = p_{i-1,r-1} - p_{i,r-1}$
    end if
    $a_{i,r} = (u - u_i)/(u_{i+p-r+1} - u_i)$
    $P_{i,r} = (1 - a_{i,r})P_{i-1,r-1} + a_{i,r}P_{i,r-1}$
  end for
end for
return $P_{k-s,p-s}$ and *normal*

- multi-Bézier-surface

- multi-B-Spline-surface with degree 3

- multi-B-Spline-surface with degree 2

- multi-B-Spline-surface with degree 1



- adaptive-mesh

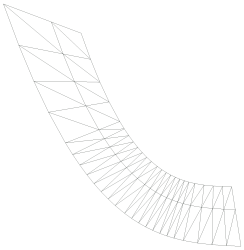


Fig. 1. adaptive mesh

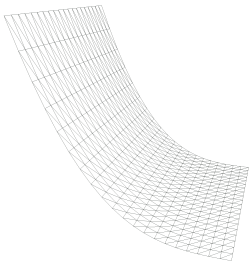


Fig. 2. uniform mesh