Assignment 6 - Visualization of parametric surfaces

Mikkel Willén

January 21, 2023

Contents

1 Introduction

This assignment is about visualization of parametric surfaces. In this text we will explain and implement algorithms for visualization of parametric surfaces. We are going to look at to different kinds of paramtric surfaces namely general parametric surfaces and Bezier surfaces, and two methods for visualization namely sampling and subdivision.

2 Theory

A paramtric surface is a surface defined by the function $Q(s,t)$. Q is a vector funktion and s and t are parameters. For every s and t value we get a 3 dimensionel vector. If we lock one of the paramters and change the other, we get a parametric curve. That can be either a Hermite curve or a Bezier curve. We can write this as:

$$Q(s,t) = \begin{cases} Q(s_0,t) \\ Q(s,t_0) \end{cases}$$

A parametric curve can be expressed by a geometric matrix, a basis matrix and a parameter vector.

$$Q(t) = GMt = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 \end{bmatrix} Mt$$

Instead of using set values as explained before, we use control points for the function. The new set of control points are:

$$Q(s,t) = G(s)Mt = \begin{bmatrix} G_1(s) & G_2(s) & G_3(s) & G_4(s) \end{bmatrix} Mt$$

These 4 function we want to be a parametric curve.

$$G_i(s) = G \begin{bmatrix} g_{ix}(s) \\ g_{iy}(s) \\ g_{iz}(s) \end{bmatrix} = \begin{bmatrix} G_{1i} & G_{2i} & G_{3i} & G_{4i} \end{bmatrix} MS$$

The control points are $\begin{bmatrix} G_{1i} & G_{2i} & G_{3i} & G_{4i} \end{bmatrix}$ which are set values. The parameter vector is now S instead of t. We can write the 4 control points as:

$$\begin{bmatrix} G_1(s) \\ G_2(s) \\ G_3(s) \\ G_4(s) \end{bmatrix} = \begin{bmatrix} G_{11} & G_{21} & G_{31} & G_{41} \\ G_{12} & G_{22} & G_{32} & G_{42} \\ G_{13} & G_{23} & G_{33} & G_{43} \\ G_{14} & G_{24} & G_{34} & G_{44} \end{bmatrix} MS$$

We need to tranpose G since we need the rows to be columns and columns to be rows.

$$\begin{bmatrix} G_1(s) & G_2(s) & G_3(s) & G_4(s) \end{bmatrix} = S^T M^T \begin{bmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \\ G_{31} & G_{32} & G_{33} & G_{34} \\ G_{41} & G_{42} & G_{43} & G_{44} \end{bmatrix}$$

We insert this in the function $Q(s,t)$ and get

$$Q(s,t) = S^T M^T \begin{bmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \\ G_{31} & G_{32} & G_{33} & G_{34} \\ G_{41} & G_{42} & G_{43} & G_{44} \end{bmatrix} Mt$$

Mt can be written as a vector, where the elements are 3rd degree polynomiums which we write as $p(t)$.

$$Mt = \begin{bmatrix} m_{11}t^3 & m_{12}t^2 & m_{13}t & m_{14} \\ m_{21}t^3 & m_{22}t^2 & m_{23}t & m_{24} \\ m_{31}t^3 & m_{32}t^2 & m_{33}t & m_{34} \\ m_{41}t^3 & m_{42}t^2 & m_{43}t & m_{44} \end{bmatrix} = \begin{bmatrix} p_1(t) \\ p_2(t) \\ p_3(t) \\ p_4(t) \end{bmatrix}$$

$S^T M^T$ will then be

$$S^T M^T = \begin{bmatrix} p_1(s) & p_2(s) & p_3(s) & p_4(s) \end{bmatrix}$$

If we insert this in $Q(s,t)$ we get

$$Q(s,t) = \begin{bmatrix} p_1(s) & p_2(s) & p_3(s) & p_4(s) \end{bmatrix} \begin{bmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \\ G_{31} & G_{32} & G_{33} & G_{34} \\ G_{41} & G_{42} & G_{43} & G_{44} \end{bmatrix} \begin{bmatrix} p_1(t) \\ p_2(t) \\ p_3(t) \\ p_4(t) \end{bmatrix}$$

The p's are 3rd degree polynomiums are defined by the basis matrix M and G is now set by the p polynomiums

For Bezier surfaces the geometric matrix G is

$$G_B = \begin{bmatrix} Q(0,0) & G_{12} & G_{13} & Q(0,1) \\ G_{21} & G_{22} & G_{23} & G_{24} \\ G_{31} & G_{32} & G_{33} & G_{34} \\ Q(1,0) & G_{42} & G_{43} & Q(1,1) \end{bmatrix}$$

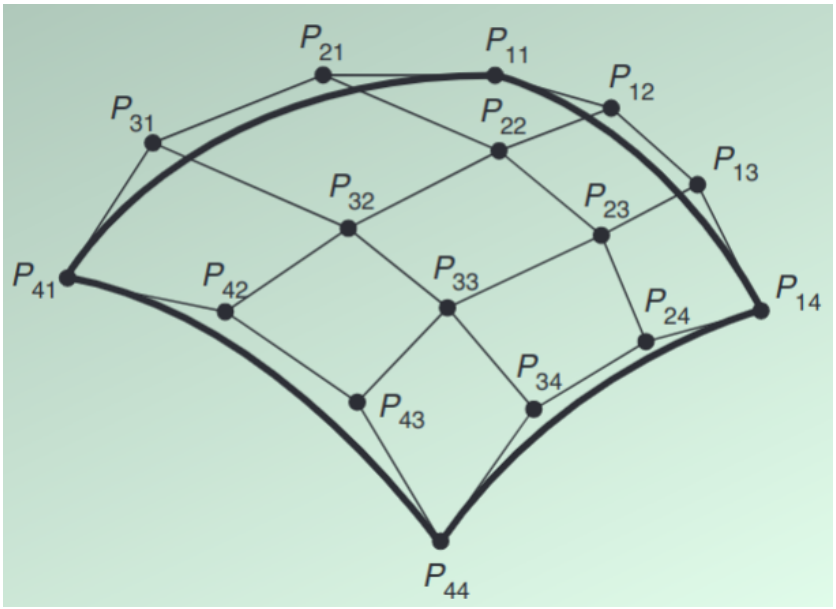

Figure 1: Bezier surface

The Q points in the geometric matrix are the corner points of the Bezier surface, the other points are control points and they lie outside the surface. This can be seen on the figure below.

The function for the Bezier surface is the same Q function we have seen before, and we have the basis matrix M_B for Bezier and the geometric matrix from above.

To visualize the surfaces we use sampling for generel parametric surfaces and sub division for Bezier surfaces. When sampling generel surfaces we calculate the points

$$f(ui, vj), \quad f(ui + u, vj), \quad f(ui + u, vj + v), \quad f(ui, vj + v)$$

These points make up a square, which we split up into 2 triangles, since openGL only work with triangles. In sampling we work with the interval $u, v) \in [u_{min}, u_{max}][v_{min}, v_{max}$. If we then want N samples for u and M samples for v we have

$$\Delta u = \frac{u_{max} - u_{min}}{N}$$
$$u_0 = u_{min}$$

$$
\begin{aligned}
u_i &= u_{i-1} + \Delta u \\
\Delta v &= \frac{v_{max} - v_{min}}{M} \\
v_0 &= v_{min} \\
v_i &= vi - 1 + \Delta v
\end{aligned}
$$

We use sub division for Bezier surfaces where we have the equation

$$
\begin{aligned}
Q(s,t) &= S^T M^T G M t \\
&= G(s) M t \\
&= \begin{bmatrix} G_1(s) & G_2(s) & G_3(s) & G_4(s) \end{bmatrix} M t
\end{aligned}
$$

and we have

$$G(s) = S^T M^T G$$

$\Leftrightarrow$

$$G(s)^T = G^T M S$$

The surface then gets split up into 2 new surfaces a left and a right, and so $s = \frac{1}{2}$. We get new control points for the surfaces DLB and DRB. We can also look at the surface from t, and we get

$$
\begin{aligned}
Q(s,t) &= S^T M^T G M t \\
&= S^T M^T G(t) \\
&= \begin{bmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{bmatrix} M t
\end{aligned}
$$

and we have

$$G(t) = GMt$$

These new surfaces can also be split up into 2 new surfaces where $t = \frac{1}{2}$. Again we get a DLB and DRB. If we multiply the points together as seen on the figure below, we get 16 control points for a new surface.
This is done recursivly so the surface is drawn. The normal of a Bezier surface is compute by the cross product of the corners.

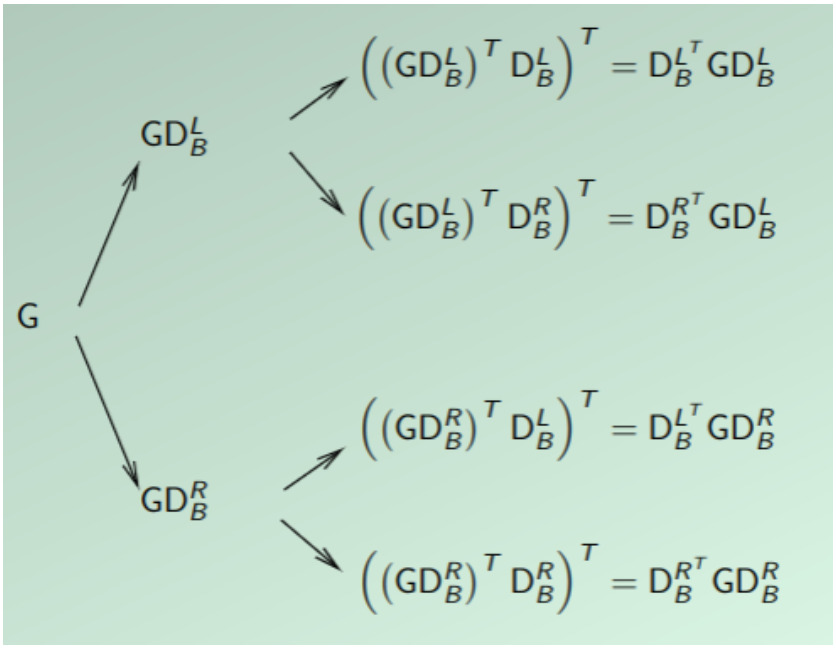

Figure 2: Sub division for Bezier surface

3 Implementation

The implementation consists of to methods for drawn parametric surfaces namely sampling and sub division. In sampling for the generel parametric surface we got the u and v. We start by defining the delta u and *delta* v. Next we make 4 points for the corners and the normal. Then we define the u parameter and define the v parameter in a loop. Again we split the square in 2 triangles. We sample then by going on step in the u direction and et step in the v direction, and make the square that way. This consists of 2 loops, one with v going from 0 to M and another with u going from 0 to N. Then we calculate the vertices and the normal coordinates. After this, we update the u and v values. We also have to statements that runs CreateFrontFracingData and CreateBackFacingData which pushes the vertices and normals. We use sampling to draw the Klein Bottle and the Dini surface.

In the implementation of sub division for the Bezier surface we start by computing the cross product of the different corners to the the normal values. After we get the 4 normal values, we check, if they are frontFacing. If they are, we push the vertices and the nromals. We do this for both triangles. If

they are not frontFacing it means we are not done dividing up the problem and that we have to make a new Bezier patch. When we have made that, we call the function recursivly with the new patch and count 1 down. We use the sub division implementation to draw the 4 figures Teapot, pain, rocket and patches. The parameters for these different figure are given.

4 Testing

We can see that by using the implemented sampling and sub division, we get the figures we wanted. The first 2 figures are drawn with sampling on generel parametric surfaces. The last 4 figures are drawn with sub division of Bezier surfaces.

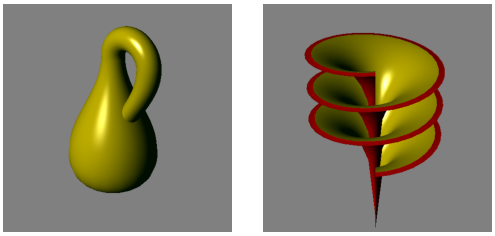

Figure 3: Tests of sampling on generel parametric surfaces

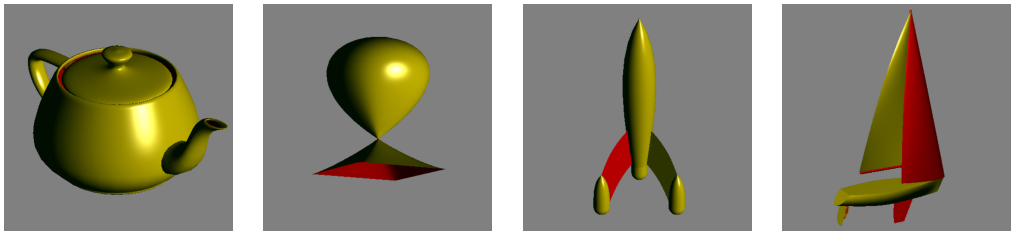

Figure 4: Tests of sub division on Bezier surfaces

5 Conclusion

We have explained and implemented algorithms for visualizing parametric surfaces. We have looked at the generel algorithm for parametric surfaces and Bezier surfaces. We have also explained and implemented sampling and sub

division for drawing the parametric surfaces. We tested the implementaion by looking at the output figures, and since they look like they should, we can conclude that the implementation works correctly.

References