A brief discussion about B-Spline, Blending Spline, Blending Spline subdivision,Blending Spline Curve shape changing and special effects with implementation

Sumaia Jahan Brinti

October 7, 2025

Abstract

The relevant theory of applied geometry with associated algorithms is extended to provide a framework for understanding and implementing B-splines curves with B function, Blending subdivision, Blending Spline Curve. The new derived polygon corresponding to an arbitrary refinement of the knot vector for an existing B-spline curve, including multiplicities, is shown to be formed by successive evaluations of the discrete B-spline defined by the original vertices, the original knot vector, and the new refined knot vector. Existing subdivision algorithms can be seen as proper special cases which leads to some different shapes of blending spline curve and it's special effects.

1 Introduction

Applied geometry is the field of study that uses the principles of geometry in finding solutions to real-life problems. Geometry allows you to determine how shapes and figures fit together to maximize efficiency and visual appeal. In daily life, we face a lot of practical situations where we need to calculate a certain distance between objects, calculate the size of a shape that needs to be attached in a position, for measurement, and much more.

2 B Spline

A B-spline function is a combination of flexible bands that is controlled by a number of points that are called control points, creating smooth curves. These functions are used to create and manage complex shapes and surfaces using a number of points. B-spline function and Bézier functions are applied extensively in shape optimization method

B-splines is basically a set of "local" basis functions that together sums up to 1 over the entire domain. Each basis function is connected to one point. The formula for a curve is,

The B-spline function of degree k with control points P_i is given by:

$$S(u) = \sum_{i=0}^{n} N_{i,k}(u) \cdot P_i$$

where u is the parameter, n is the number of control points, and $N_{i,k}(u)$ is the i-th B-spline basis function of degree k evaluated at u. The basis functions are defined recursively as:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(u) = \frac{u - u_i}{u_{i+k} - u_i} N_{i,k-1}(u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} N_{i+1,k-1}(u)$$

where u_i is the i-th knot value.

2.1 B-Spline Curve

In the B-spline curve, the control points impart local control over the curve-shape rather than the global control like Bezier-curve.

B-spline curve shape before changing the position of control point P1 –

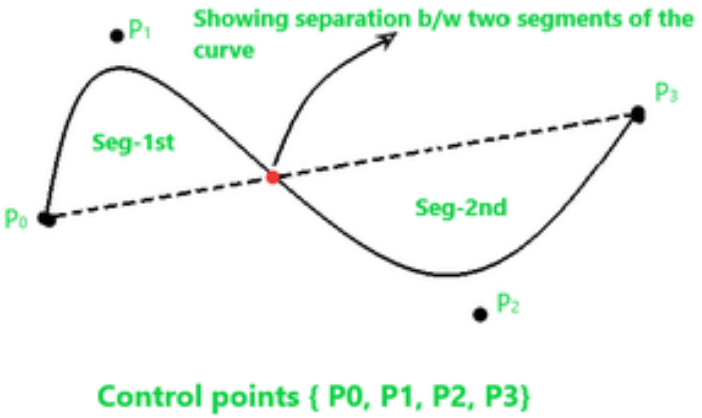


Figure 2.1: Control Point P0, P1, P2, P3

B-splines curves are independent of the number of control points and made up of joining the several segments smoothly, where each segment shape is decided by some specific control points that come in that region of segment. Consider a curve given below –

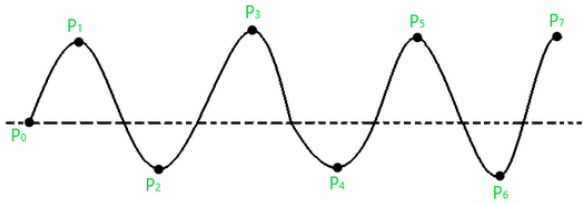


Figure 2.2: B-splines curve

Properties of B-spline Curve :

- Each basis function has 0 or +ve value for all parameters.
- Each basis function has one maximum value except for k=1.
- The degree of B-spline curve polynomial does not depend on the number of control points which makes it more reliable to use than Bezier curve.
- B-spline curve provides the local control through control points over each segment of the curve.
- The sum of basis functions for a given parameter is one.[**?**]

2.2 Knot Vector

The set of knot values that, together with a polynomial degree d, defines the spline space is denoted by the knot vector, $t = t_0, t_1, \ldots, t_{n+d}$. The B-spline functions, which form a basis for this spline space, are defined recursively using Cox-de Boor formula.

The partition of unity property requires that the sum of the basic functions must be equal to 1 over the entire domain of the spline. Therefore, there must be $d + 1$ active basic functions at all knot intervals.

For a given knot vector $t = t_0, t_1, \ldots, t_{n+d}$, the domain of any B-spline function/curve of degree d based on this knot vector restricted to $[t_d, t_n]$.

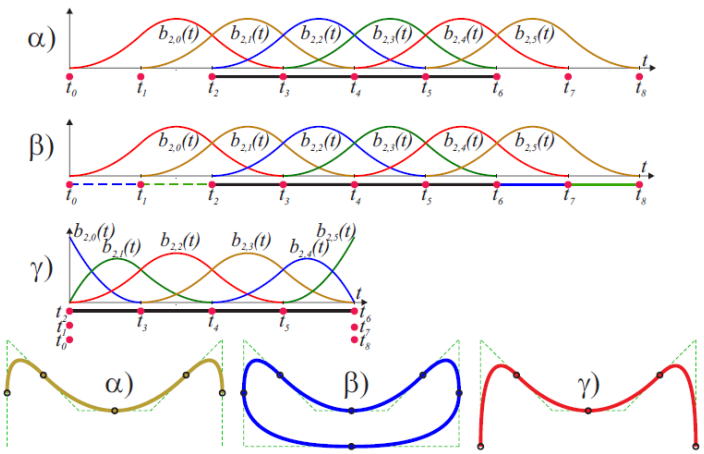


Figure 2.3: A B-spline function / curve can have three states determined by the knot vector, open, clamped or closed/cyclic.

A B-spline function / curve can have three states determined by the knot vector, open, clamped or closed/cyclic.

2.3 B-spline curves - Open, Clamped or Closed

B-spline curves share many properties with Bézier curves, because the former is a generalization of the latter. In the following, we denote a B-spline curve c(t). It is defined by a polynomial degree d, and thus order k = d+1, and by a knot vector t = t0, t1, ..., tn+d and n control points. The general formula is

$$c(t) = \sum_{i=0}^{n-1} c_i b_{k,i}(t), \tag{1}$$

where c_i is the vector of coefficients/control points. These control points define the control polygon of the curve. $b_{k,i}(t)$ is the set of B-spline basis functions defined by the knot vector t, which has n+k knot values.

As mentioned, a B-spline curve can have three states: open, open/clamped, or closed/cyclic. If a curve is constructed from n basis functions and thus n control points, then

Open - is a curve that is built on an ordinary set of knot values $t = t0, t1, ..., tn + d$ where $t_{i+1} \geq t_i$ and where the domain of the curve is restricted to $[t_d, t_n]$.

Open/Clamped - is an open curve with a knot vector $t = t0, t1, ..., tn + d$, but where the first k and last k knots are equal, i.e., $t_0 = t_1 = \cdots = t_d$ and $t_n = t_{n+1} = \cdots = t_{n+d}$.

Closed/Cyclic - is a curve where the knot vector "bites itself in the tail", it has a knot vector t = t0, t1, ..., tn+d, but the domain is extended to $[t_d, t_{n+d})$. In addition, $t_{n+i+1} - t_{n+i} = t_{i+1} - t_i$, $i = 0, 1, \ldots, d-1$.

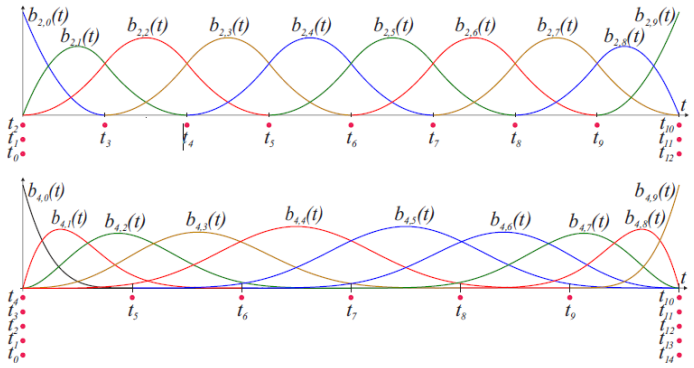


Figure 2.4: At the top we see a set of 10 2^{nd}-degree B-spline (basis) functions b2;i(t), defined by a knot vector of 13 knots, marked with red bullets, and at the bottom we see a set of 10 4^{th}-degree B-spline (basis) functions b4;i(t), defined by a knot vector of 15 knots.

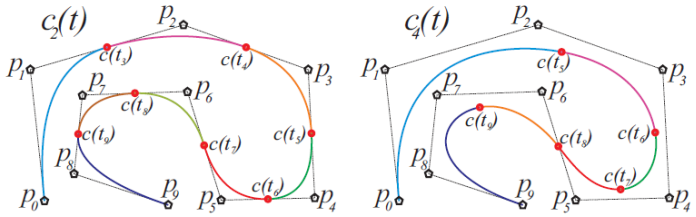


Figure 2.5: A 2^{nd}-degree B-spline curves c2(t) and a 4^{th}-degree B-spline curve c4(t) based on the same control points. The red dots mark position of internal knot values

Here , we see the B-splines based on these two knot vectors. In previous Figure , B-spline curves based on these two set of basis functions plotted. We see that the 2^{nd}-degree curve touches the control polygon due to that only two basis function are 0 at the knot values where n is the number of control points and $b_i^j(t)$ is the i-th basis function of degree j.

A B-spline curve is polynomial-based curve pieces that are glued together at the knot values. Therefor, for a 2^{nd}-degree $[t_i, t_{i+1})$, where $i = d, d+1, \ldots, n$, we get the following formula (provided that $t_{i+1} > t_i$):

$$c(t) = \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1 - w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}$$

We call these matrices for factor matrices because they actually factorize the B-splines over one knot interval.

2.4 B-functions

B-function is an abbreviation for blending function. It is for blending functions, whether they are based on scalar-, vector- or point-values, such as points, curves, tensor product or triangular surfaces etc. In the following, we restrict B functions to be monotonous and we will look especially at symmetric B-functions and what it means. The B-function is:

D1 a homeomorphism ('permutation function') $B : [I] \to [0, 1] \subset R$,

D2 such that $B(0) = 0$,

D3 such that $B(1) = 1$,

D4 and that is monotone, i.e., $B'(t)$ 0 for $t \in [0, 1]$,

D5 A B-function is called symmetric if $B(t) + B(1 - t) = 1$ for $t \in [0, 1]$.

To give an idea of what a B-function is, we will look at four simple examples of symmetric B-functions:

a) linear function $B(t) = t$

b) trigonometric function $B(t) = \sin^2(\pi t)$

c) polynomial function of first order $B(t) = 3t^2 - 2t^3$

d) rational function of first order $B(t) = \frac{t^2}{t^2+(1-t)^2}$

2.5 Implementation of B spline

Considering the theory, we have implemented open B-Spline curve with knot vectors and findControlPoints ,there we used two constructors (fig-6). Here we have used multiple constructors since we wanted to input varying amounts of initial data for an object. At first case, we have initialized the control points and knot vector with dimension, In second constructor we called the control points.In eval function , we called findI() and with that index called getB() for second degree blending function to compute the position and d derivatives of the curve.

```
template <typename T>
inline
MyB_spline<T>::MyB_spline(const DVector<Vector<T,3>>& c):PCurve<T,3>(20, 0, 0), _d(2), _k(3) {

  _c = c;
  makeKnots(c.getDim(), _k);
}


template <typename T>
inline
MyB_spline<T>::MyB_spline(const DVector<Vector<T,3>>& p, int n):PCurve<T,3>(20, 0, 0), _d(2), _

  _c.setDim(n);
  makeKnots(n, _k);

   // Find control points
  int m = p.getDim();
  findControlPoints(p,m,n);
}
```

Figure 2.6: Constructors of B-splines curve

The output of B-splines is

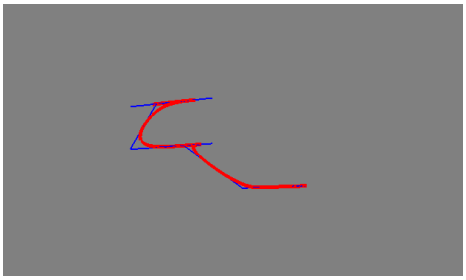


Figure 2.7: B-spline

2.6 Implementation of Blending-spline

Considering the thoery, we have constructed Blending-Spline curve with knot vectors open and closes and initilazed it with a model curve equation. Images of model curve and the constructor of Blending splines are respectively below

```
this->_p[0][0] = a * cos(Kx * t);
this->_p[0][1] = b * sin(Ky * t);
this->_p[0][2] = T(0);

if( this->_dm == GM_DERIVATION_EXPLICIT ) {
  if( d > 0 ) {
    this->_p[1][0] = -a* sin(Kx * t);
    this->_p[1][1] = b* cos(Ky * t);
    this->_p[1][2] =  T(0);
  }
```

(a) model curve

```
template <typename T>
inline
MyBlending<T>::MyBlending(PCurve <T ,3 >* model_curve, int n):PCurve<T,3>(20, 0, 0), _d(1), _k(2)

    _modelCurve = model_curve;
    _n=n;

    if(isClosed())
    {
        _c.setDim(n+1);
        makeKnotsClosed(n, _modelCurve->getParStart(),_modelCurve->getParEnd());
        for(int i=0;i<n; i++)
        {
            _c[i]= new PSubCurve<T>(_modelCurve,_t[i],_t[i+2],_t[i+1]);
            _c[i]->toggleDefaultVisualizer();
            _c[i]->sample(10,0);
            this->insert(_c[i]);
            _c[i]->setCollapsed(true);
        }
        _c[n]=_c[0];
    }
    else
    {
        _c.setDim(n);
        makeKnotsOpen(n, _modelCurve->getParStart(),_modelCurve->getParEnd());
        for(int i=0;i<n; i++)
        {
            _c[i]= new PSubCurve<T>(_modelCurve,_t[i],_t[i+2],_t[i+1]);
            _c[i]->toggleDefaultVisualizer();
            _c[i]->sample(10,0);
```

(b) constructor

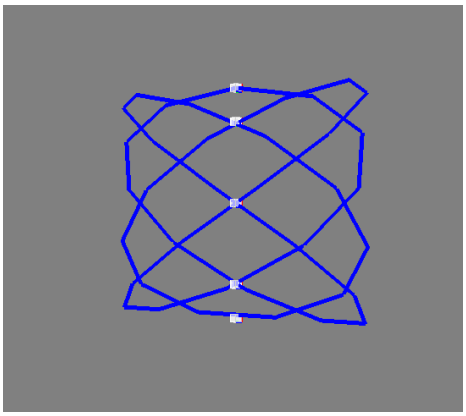


Figure 2.9: Blending spline

In ***eval*** method , we are considering two one for knot vector open and another for closed that's why evaluatiing c1 and c2. Using localcurves, I have used localSimulate() to simulate this one. We have used the blend function and

3 Subdivision Surfaces

Subdivision surfaces are mostly used in Computer Graphics and virtual design. The subdivision surfaces algorithm is recursive in nature. The starting point for a subdivision algorithm is a set of points connected in a network that covers and defines an underlying surface. The data must be arranged so that the inside and outside can be determined locally in each polygon. There are ready-made data structures for programming subdivision, such as OpenMesh. but it is also possible to use a simple self-maid data structure. For example, arrays of three types of objects, a Vertex (a point and a surface normal), a Face (a polygon - i.e an ordered set of Vertex indices,organized counterclockwise seen from the outside) and an edge (two Vertex indices plus two Face indices). NB! When implementing, it is important, and also possible, to make all parts of an algorithm so that they are of $O(n)$. If some parts are of $O(nlogn)$ or $O(n2)$ the subdivision will be very slow when the number of vertices becomes large. The classifications based on geometric properties are mainly:

- whether they are corner cutting (i.e approximating) or interpolating schemes,
- the continuity, i.e C1-smooth or C2-smooth,
- whether they mainly are based on triangles or quads.

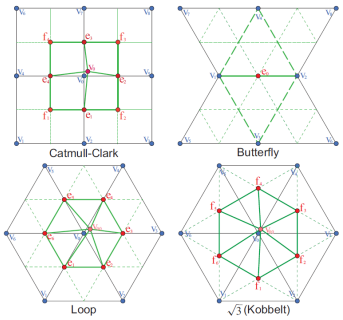


Figure 3.10: Selection of Subdidvision

3.1 Algorithm

Algorithm 1 Lane-Riesenfeld Algorithm

```
1: procedure VECTOR< Point > LaneRiesenfeldClosed(VECTOR< Point > P, INT k, INT d)( )
2:     int n = P.size —                          ▷ The number of intervals
3:     int m = 2kn + 1;                          ▷ The final number of points
4:     vector < Point > F(m);                    ▷ The return vector – m points.
5:     for i ← length(input) − 0 to n do
6:         Fi = Pi;                              ▷ Inserting the initial points
7:     end for
8:     Fn = P0;                                  ▷ Closing the curve
9:     for i ← length(input) − 0 to k do
10:        n = doublePart(F, n)
11:       smoothPartClosed(F, n, d);
12:    end for
13:    return F;
14: end procedure
```

Algorithm 2

```
1: procedure INT doublePart(VECTOR< Point > P, INT n)( )
2:     for i ← n − 1 to 0 ; i − −; do
3:         P2i = Pi;
4:         P[2i − 1] = 1/2(Pi + P[i − 1]);
5:     end for
6:     return 2n − 1;
7: end procedure
```

Algorithm 3

1: **procedure** VOID SMOOTHPARTCLOSED((VECTOR< $Point >$ P, INT n, INT d)( )
2: **for** $j \leftarrow 1$ to d ; $j++$; **do**
3: **for** $j \leftarrow 1$ to d ; $j++$; **do**
4: $P_[i] = 1/2(P_i + P_[i+1])$;
5: **end for**
6: $P_[n-1] = P_0$
7: **end for**
8: **end procedure**

3.2 Arguement for Model Curve

There are several arguments for the choice of B-spline curves as a model curve, including:

Flexibility: B-spline curves are highly flexible and can be used to model a wide range of shapes and curves. The shape of the curve can be easily adjusted by manipulating the control points, allowing for greater control over the shape and curvature of the curve.

Smoothness: B-spline curves are known for their smoothness and continuity properties. The degree of the curve can be adjusted to control the degree of smoothness, with higher degrees resulting in smoother curves. The use of B-spline curves can result in more aesthetically pleasing designs and animations.

Efficiency: B-spline curves are computationally efficient and can be evaluated quickly and accurately. This makes them ideal for use in computer graphics and design applications where real-time rendering and responsiveness are important.

Interpolation: B-spline curves can be used for curve interpolation, which means that the curve can be made to pass through a set of specified points. This is useful in situations where precise control over the shape of the curve is required, such as in design applications.

3.3 Implementation of subdivision

Considering the theory, we have implemented local curve which used LaneRiesenfieldClosed algorithm along with doublePart() and smoothPartClosed() and computed Surrounding Sphere

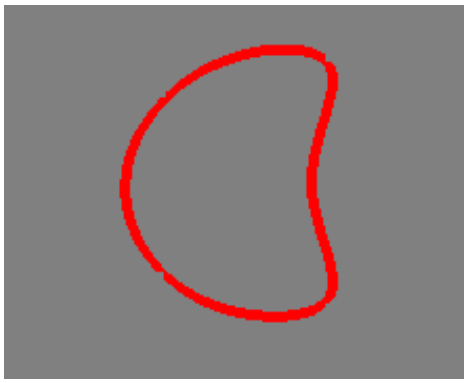


Figure 3.11: subdivision

3.4 Implementation of Blending Surface

We have implemented A constructor for initialization of model surface start parameter for u direction an v direction knot-factor definition if curve is closed or open and in u and v direction.We computed blend function and it's derivative in u an v direction and then evaluates degree in both direction to get position.For this case I have used cylinder to support the surface.

```
if(isClosedV())
    nv++;
if (isClosedU())
    makeKnotsClosed(_u, _nu, p->getParStartU(), p->getParEndU() );
else
    makeKnotsOpen(_u, _nu, p->getParStartU(), p->getParEndU() );
if (isClosedV())
    makeKnotsClosed(_v, _nv, p->getParStartV(), p->getParEndV() );
else
    makeKnotsOpen(_v, _nv, p->getParStartV(), p->getParEndV() );

_c.setDim(nu, nv);

for(int i= 0; i<_nu;i++){
    for(int j= 0; j<_nv;j++){
        _c[i][j] = new PSimpleSubSurf<T>(_modelsurface, _u[i], _u[i+2], _u[i+1], _v[j], _
        _c[i][j]->toggleDefaultVisualizer();
        _c[i][j]->sample(10,10,1,1);
        this->insert(_c[i][j]);
        _c[i][j]->setCollapsed(true);
    }

    if (isClosedU())
      _c[_nu][i] = _c[0][i];
}
if (isClosedV())
{
    for (int i =0; i<_nu; i++)
      _c[i][_nv] = _c[i][0];
}
if (isClosedV() && isClosedU())
```

Figure 3.12: Blending Surface

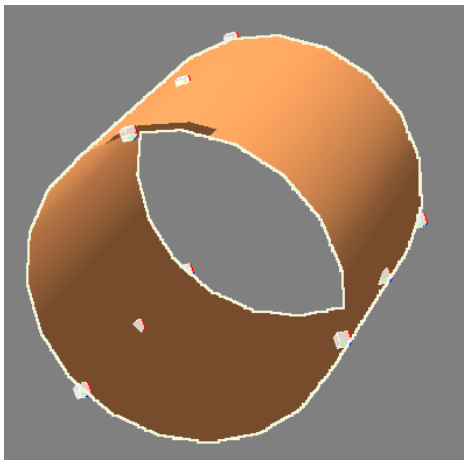


Figure 3.13: Blending Surface

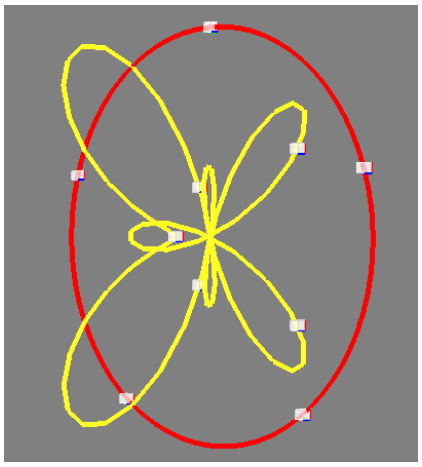
3.5 Implementation of Blending spline curve

Based on blending spline I have taken another curve where I implemented parametric equation and in my blending spline class I have used bfunction and rotateParent and translateParent for simulating my butterfly and petals.I have used two curve in Scenario.cpp to make a butterfly into a outer circle.

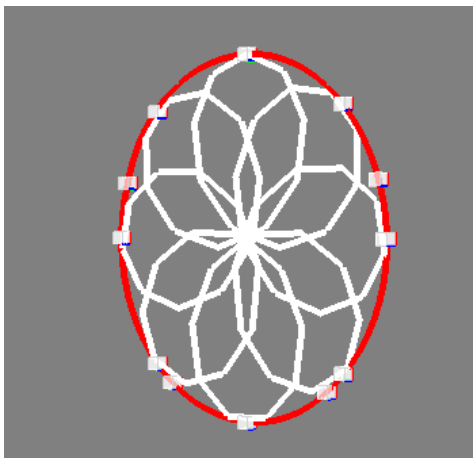
```
//     this->_p[0][2] = T(0);
//rose petal equation
this->_p[0][0] = a * cos(k*t)*cos(3*t);
this->_p[0][1] = a * cos(k*t)*sin(3*t);
this->_p[0][2] = T(0);

// buterfly equation
//    this->_p[0][0] = sin(t) * ((exp(cos(t))) - 2 * cos(4*t) - pow(sin(t/12),5));
//    this->_p[0][1] = cos(t) * ((exp(cos(t))) - 2 * cos(4*t) - pow(sin(t/12),5));
//    this->_p[0][2] = T(0);
```

Figure 3.14: Parametric Equation



(a) Butterfly



(b) Rose

3.6 Special effects

Here we have used a counter and three direction to translate and rotate. During this translation butterfly/rose changes its shape and then again come to it's original shape with changing color.

4 Conclusion

In this article we have discussed about B-Spline, Knot vector, B function, Subdivision, surfaces, parametric equation theorem and their implementation with their results.

5 Appendix Project Setup

We need to clone the repository and need to install Qt necessary components. To execute the program, you can locate the repository and files. Next, clone the project and store it in a folder next to the GMlib library. Then, create a build folder and within it, make two subfolders named after the GMlib library and the project. Both subfolders should contain the building directory for their respective projects. It's important to remember that after running the CMake file for the project, the GMlib library should be included as a project dependency.

There is the scenario.cpp file located within the project folder that contains various scenarios that can be uncommented and executed. It is advisable to activate one scenario at a time to ensure optimal performance and good result. Repository link is https://source.coderefinery.org/sbr132/appliedgeometry/-/tree/main

References

[1] camac. bspline. https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node4.html).

[2] Geeksforgeeks. Geeksforgeeksbspline. https://www.geeksforgeeks.org/b-spline-curve-in-computer-graphics/).

[3] Arne Laksa. *Blending Techniques in Curve and Surface Construction.* CCBY, N-8505 Narvik, Norway, 1st edition, 2022.

[4] mtu. Bsplinecurve. https://pages.mtu.edu/ shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-curve.html.

[5] stanford. Warrensubdivision. http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/warren-subdivision.pdf).

[6] Sumaia. Flowerbuterflyproject. https://source.coderefinery.org/sbr132/appliedgeometry/-/tree/main.

[7] UIO. Basicbspline. https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF-MAT5340/v05/undervisningsmateriale/kap2-new.pdf).

[8] wikipedia. Bspline. https://en.wikipedia.org/wiki/B-spline.

[9] wikipedia. Petalareas. https://en.wikipedia.org/wiki/Rose$_(mathematics)Total_and_petal_areas$).

[10] wikipedia. Rosecurve. https://en.wikipedia.org/wiki/Rose$_(mathematics)$.

[6] [3] [8] [4] [10] [9] [7] [1] [5] [2]