به نام خدا

گزارش پروژه درس بینایی ماشین

رسم منحنی های بسته با روش Bezier Spline

استاد: دکتر میرهادی سید عربی

ارائه دهنده: امیر محسن یوسفی واقف

تیرماه ۱۳۹۰

فهرست

مقدمه

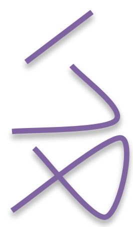
در زمینه ریاضیات آنالیز عددی و در گرافیک کامپیوتری یک Bezier Spline یک منحنی Spline می باشد که هر چند جمله ای از Spline در فرم Bezier تعریف شده است.

به تعریف دیگر Bezier Spline به سادگی یک سری از منحنی های Bezier می باشد که پایان آنها به هم متصل شده است، جایی که آخرین نقطه از یک منحنی همزمان با نقطه شروع منحنی بعدی می باشد، معمولاً منحنی های مکعبی Bezier استفاده می شوند و نقاط کنترلی برای تعریف شکل هر منحنی استفاده می شود.

Bezier Curve

انواع توابع چند جمله ای عبارتند از:

- خطی $f(t)=at+b$
- مربعی $f(t)=at^2+bt+c$
- مکعبی $f(t)=at^3+bt^2+ct+d$

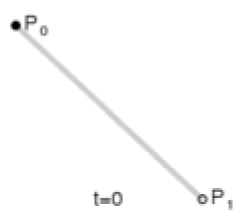


We usually define the curve for $0 \leq t \leq 1$

منحنی خطی Bezier

با توجه به P0, P1 منحنی Bezier خط راستی است که از دو نقطه می گذرد:

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , \ t \in [0,1]$$



شکل ۱ -منحنی خطی Bezier

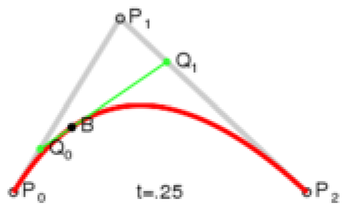
منحنی درجه دوم Bezier

منحنی درجه دوم Bezier مسیر تابع B(t) را بین نقاط P_0, P_1, P_2 پیمایش میکند که می تواند همچون خطی تفسیر شود که مطابق یا منحنی Bezier خطی از P_0 به P_1 و از P_1 به P_2 درون یابی می شود:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \ , \ t \in [0,1].$$

برای منحنی Bezier درجه دوم ما می توانیم نقاطی میانی مانند Q_0, Q_1 که t از ۰ تا۱ تغییر میکند:

- نقطه Q_0 از P_0 تا P_1 تغییر می کند و یک منحنی Bezier را تعریف میکند.
- نقطه Q_1 از P_1 تا P_2 تغییر می کند و یک منحنی Bezier را تعریف میکند.
- نقطه B(t) از Q_0 تا Q_1 تغییر می کند و منحنی درجه دوم Bezier را تعریف میکند.



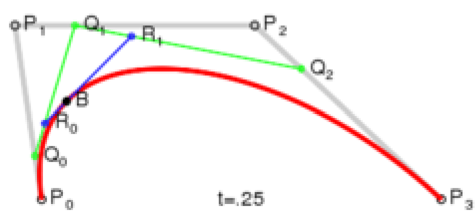
شکل ۲ -منحنی درجه دوم Bezier

منحنی مکعبی Bezier

هنگامی که چهار نقطه P_0, P_1, P_2, P_3 در فضای سه بعدی تعریف شود منحنی مکعبی Bezier را تشکیل می دهد. به طوری که از نقاط P_1, P_2 منحنی نمی گذرد و فقط اطلاعات مسیر را فراهم می آورد:

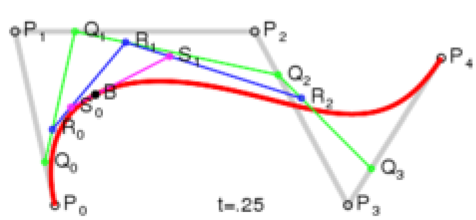
$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , \ t \in [0,1].$$

برای منحنی Bezier مکعبی ما سه نقطه Q_0, Q_1, Q_2 را به عنوان درون یابی مد نظر گرفته و نقاط R_0, R_1 منحنی Bezier مربعی را توصیف می کند.



شکل ۳ -منحنی مکعبی Bezier

برای منحنی های درجه بالاتر کافی است نقاط درون یابی و نقاطی که منحنی Bezier مربعی را تشکیل میدهند بیشتر نمائیم.



شکل ۴ -منحنی درجه ۴ Bezier

برای تعریف بازگشتی ما فرم کلی زیر را در نظر می گیریم

$$\mathbf{B}(t) = \mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_n}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_{n-1}}(t) + t\mathbf{B}_{\mathbf{P}_1\mathbf{P}_2\ldots\mathbf{P}_n}(t)$$

و برای تعریف صریح منحنی Bezier فرمول زیر را داریم

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i}(1-t)^{n-i}t^i\mathbf{P}_i \\ &= (1-t)^n\mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1}t\mathbf{P}_1 + \cdots \\ &\cdots + \binom{n}{n-1}(1-t)t^{n-1}\mathbf{P}_{n-1} + t^n\mathbf{P}_n, \quad t \in [0,1],\end{aligned}$$

Bezier Spline

یک Spline بنام S با درجه n و k تا گره (knot) x_i می تواند یک Spline را به صورت Bezier Spline تعریف کند

$$S(x) := \begin{cases} S_0(x) := & \sum_{\nu=0}^{n} \beta_{\nu,0} b_{\nu,n}(x) & x \in [x_0, x_1) \\ S_1(x) := & \sum_{\nu=0}^{n} \beta_{\nu,1} b_{\nu,n}(x - x_1) & x \in [x_1, x_2) \\ \vdots & \vdots & \\ S_{k-2}(x) := & \sum_{\nu=0}^{n} \beta_{\nu,k-2} b_{\nu,n}(x - x_{k-2}) & x \in [x_{k-2}, x_{k-1}] \end{cases}$$

تقریب کمان های دایره

هرگاه کمانهای اولیه دابری در محیط مخصوصی پشتیبانی نمی شود آنها ممکن است بوسیله منحنی های Bezier تخمین زده شوند و بطور عموم چهارتا تقسیمات مربعی برای تخمین دایره ممکن است استفاده شود. مطلوب است پیدا کردن K نقطه کنترلی که حداقل خطای تخمین را نتیجه بدهد.

استفاده از ۴ منحنی:

برای کمان ربع یک چهارم دایره با نقاط پایانی A و B و نقاط کنترلی A’ و B’ به صورت زیر تعریف می کنیم:

$$\mathbf{A} = [0,1]$$
$$\mathbf{A}' = [\mathbf{k},1]$$
$$\mathbf{B}' = [1,\mathbf{k}]$$
$$\mathbf{B} = [1,0]$$

از تعریف منحنی Bezier مکعبی داریم که:

$$\mathbf{C}(t) = (1-t)^3\mathbf{A} + 3(1-t)^2 t\mathbf{A}' + 3(1-t)t^2\mathbf{B}' + t^3\mathbf{B}$$

با تعریف نقطه c(t=0.5) همچون نقطه میانی کمان ما دو معادله زیر بدست می آید:

$$\mathbf{C} = \frac{1}{8}\mathbf{A} + \frac{3}{8}\mathbf{A}' + \frac{3}{8}\mathbf{B}' + \frac{1}{8}\mathbf{B}$$
$$\mathbf{C} = \sqrt{1/2} = \sqrt{2}/2$$

با حل این دو معادله برای محور xها داریم:

$$\frac{0}{8}+\frac{3}{8}\mathbf{k} + \frac{3}{8} + \frac{1}{8} = \sqrt{2}/2$$

$$\mathbf{k} = \frac{4}{3}(\sqrt{2}-1) = 0.5522847498$$

توصیف کدهای پروژه

ابتدا متد Slove از کلاس Cyclic را به صورت زیر پیاده سازی می نمائیم:[1]

```
public static class Cyclic
        {
            public static double[] Solve(double[] a, double[] b, double[] c, double alpha, double beta, double[] rhs)
                {
                        // a, b, c and rhs vectors must have the same size.
                        if (a.Length != b.Length || c.Length != b.Length || rhs.Length != b.Length)
                                throw new ArgumentException("Diagonal and rhs vectors must have the same size.");
                        int n = b.Length;
                        if (n <= 2)
                                throw new ArgumentException("n too small in Cyclic; must be greater than 2.");

                        double gamma = -b[0]; // Avoid subtraction error in forming bb[0].
                        // Set up the diagonal of the modified tridiagonal system.
                        double[] bb = new Double[n];
                        bb[0] = b[0] - gamma;
                        bb[n-1] = b[n - 1] - alpha * beta / gamma;
                        for (int i = 1; i < n - 1; ++i)
```

[1] - این متد از روی کتاب "Numerical Recipes in C", Chapter 2.4 "Tridiagonal and Band Diagonal Systems of Equations" پیاده سازی شده است.

```
                        bb[i] = b[i];
                // Solve A · x = rhs.
                double[] solution = Tridiagonal.Solve(a, bb, c, rhs);
                double[] x = new Double[n];
                for (int k = 0; k < n; ++k)
                        x[k] = solution[k];

                // Set up the vector u.
                double[] u = new Double[n];
                u[0] = gamma;
                u[n-1] = alpha;
                for (int i = 1; i < n - 1; ++i)
                        u[i] = 0.0;
                // Solve A · z = u.
                solution = Tridiagonal.Solve(a, bb, c, u);
                double[] z = new Double[n];
                for (int k = 0; k < n; ++k)
                        z[k] = solution[k];

                // Form v · x/(1 + v · z).
                double fact = (x[0] + beta * x[n - 1] / gamma)
                        / (1.0 + z[0] + beta * z[n - 1] / gamma);

                // Now get the solution vector x.
                for (int i = 0; i < n; ++i)
                        x[i] -= fact * z[i];
                return x;
        }
    }
}
```

سپس متد Slove از کلاس Tridiagonal را به صورت زیر پیاده سازی می کنیم:[۲]

```
namespace NumericalRecipes
{
    /// <summary>
    /// Tridiagonal system solution.
    /// </summary>
    public static class Tridiagonal
    {
        /// <summary>
        /// Solves a tridiagonal system.
        /// </summary>
        /// <remarks>
        /// All vectors have size of n although some elements are not used.
        /// </remarks>
        /// <param name="a">Lower diagonal vector; a[0] not used.</param>
        /// <param name="b">Main diagonal vector.</param>
        /// <param name="c">Upper diagonal vector; c[n-1] not used.</param>
        /// <param name="rhs">Right hand side vector</param>
        /// <returns>system solution vector</returns>
        public static double[] Solve(double[] a, double[] b, double[] c, double[] rhs)
        {
                // a, b, c and rhs vectors must have the same size.
                if (a.Length != b.Length || c.Length != b.Length || rhs.Length != b.Length)
                        throw new ArgumentException("Diagonal and rhs vectors must have the same size.");
                if (b[0] == 0.0)
                        throw new InvalidOperationException("Singular matrix.");
    // If this happens then you should rewrite your equations as a set of
                // order N - 1, with u2 trivially eliminated.

             ulong n = Convert.ToUInt64(rhs.Length);
                double[] u = new Double[n];
    double[] gam = new Double[n]; // One vector of workspace, gam is needed.

                double bet = b[0];
                u[0] = rhs[0] / bet;
                for (ulong j = 1;j < n;j++) // Decomposition and forward substitution.
                {
                    gam[j] = c[j-1] / bet;
```

[۲] - این متد از روی کتاب "Numerical Recipes in C", Chapter 2.4 "Tridiagonal and Band Diagonal Systems of Equations" پیاده سازی شده است.

```
            bet = b[j] - a[j] * gam[j];
                        if (bet == 0.0)
                                // Algorithm fails.
                                throw new InvalidOperationException("Singular matrix.");
            u[j] = (rhs[j] - a[j] * u[j - 1]) / bet;
        }
        for (ulong j = 1;j < n;j++)
                        u[n - j - 1] -= gam[n - j] * u[n - j]; // Backsubstitution.

                return u;
            }
        }
}
```

سپس ما تابع GetCurveControlPoint را در کلاس ClosedBezierSpline داریم که نقاط کنترلی را با توجه به تعداد نقاط(توسط Slider در اجرای برنامه) و مختصات آنها محاسبه می کند:

```
public static class ClosedBezierSpline
        {
                /// <summary>
                /// Get Closed Bezier Spline Control Points.
                /// </summary>
                /// <param name="knots">Input Knot Bezier spline points.</param>
                /// <param name="firstControlPoints">Output First Control points array of the same
                /// length as the <paramref name="knots"/> array.</param>
                /// <param name="secondControlPoints">Output Second Control points array of of the same
                /// length as the <paramref name="knots"/> array.</param>
                public static void GetCurveControlPoints(Point[] knots, out Point[] firstControlPoints, out Point[]
secondControlPoints)
                {
                        int n = knots.Length;
                        if (n <= 2)
                        {
                                firstControlPoints = new Point[0];
                                secondControlPoints = new Point[0];
                                return;
                        }

                        // Calculate first Bezier control points

                        // The matrix.
                        double[] a = new double[n], b = new double[n], c = new double[n];
                        for (int i = 0; i < n; ++i)
                        {
                                a[i] = 1;
                                b[i] = 4;
                                c[i] = 1;
                        }

                        // Right hand side vector for points X coordinates.
                        double[] rhs = new double[n];
                        for (int i = 0; i < n; ++i)
                        {
                                int j = (i == n - 1) ? 0 : i + 1;
                                rhs[i] = 4 * knots[i].X + 2 * knots[j].X;
                        }
                        // Solve the system for X.
                        double[] x = Cyclic.Solve(a, b, c, 1, 1, rhs);

                        // Right hand side vector for points Y coordinates.
                        for (int i = 0; i < n; ++i)
                        {
                                int j = (i == n - 1) ? 0 : i + 1;
                                rhs[i] = 4 * knots[i].Y + 2 * knots[j].Y;
                        }
                        // Solve the system for Y.
                        double[] y = Cyclic.Solve(a, b, c, 1, 1, rhs);

                        // Fill output arrays.
                        firstControlPoints = new Point[n];
                        secondControlPoints = new Point[n];
```

```
            for (int i = 0; i < n; ++i)
            {
                // First control point.
                firstControlPoints[i] = new Point(x[i], y[i]);
                // Second control point.
                secondControlPoints[i] = new Point(2 * knots[i].X - x[i], 2 * knots[i].Y - y[i]);
            }
        }
    }
```

بقیه کدهای برنامه کارهای گرافیکی و اجرایی برنامه می باشد.

مراجع

1- http://en.wikipedia.org/wiki/B%C3%A9zier_curve
2- http://en.wikipedia.org/wiki/B%C3%A9zier_spline
3- APPROXIMATION OF A CUBIC BEZIER CURVE BY CIRCULAR ARCS AND VICE VERSA – By Aleksas Riškus