1 曲线

1.1 曲线数学基础

1.1.1 曲线的表示

一般而言, 我们可以把曲线 (以及以后的曲面) 的表示方式分为显式表示和隐式表示.

定义 1.1 显式表示 显式表示是可以直接通过表达式得到点的表示方式. 例如, 平面上的圆的参数方程即显示表示:

$$\begin{cases} x = r\cos t \\ y = r\sin t \end{cases} \quad t \in [0, 2\pi)$$

一般的二次曲线也是显示表示:

$$y = ax^2 + bx + c$$

定义 1.2 隐式表示 隐式表示是指通过隐式方程来表示曲线 (或曲面), 而不直接给出参数到坐标的映射的表示方法. 例如, 平面上的圆可以表示为隐式方程

$$f(x, y) = x^2 + y^2 - r^2 = 0$$

隐式表示可以更容易地分辨曲线的内外侧, 但相应地不容易直接得到曲线的形状. 因此, 在曲线绘制时更常用显式表示.

1.1.2 曲线插值与基函数

定义 1.3 基函数 给定 $n+1$ 个点 $(x_0, y_0), \cdots, (x_n, y_n)$, 如果存在 $n+1$ 个函数 $\phi_0(x), \cdots, \phi_n(x)$, 使得

$$\phi_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

那么称 $\phi_i (i = 0, \cdots, n)$ 为这些点的**基函数 (Basis Function)**. 对这些点插值的结果可以写作

$$y = \sum_{i=0}^{n} y_i \phi_i(x)$$

这恰好是过各点的曲线, 具体形状由基函数的性质决定.

1.1.3 曲线的连续性

定义 1.4 参数连续性 设曲线的 n 阶导数在连接的两段曲线的交点处相等, 则称这两段曲线在该点处是 C^n 连续的.

定义 1.5 几何连续性 n 阶几何连续的定义如下:

1. G^0 连续: 两段曲线在连接点处相交.
2. G^1 连续: 两段曲线在连接点处相交, 且切线方向相同, 即曲率方向相同而大小不同.
3. G^2 连续: 两段曲线在连接点处相交, 且曲率方向和大小均相同.
4. G^3 连续: 两段曲线在连接点处相交, 且曲率方向, 大小和变化率均相同.

1.2 Bezier 曲线

1.2.1 Bezier 曲线的定义

Bezier 曲线是计算机图形学中常用的一种参数曲线, 由法国工程师 Pierre Bézier 在 20 世纪 60 年代为汽车车身设计而开发. 它们广泛应用于计算机图形学、动画、字体设计等领域.

Bezier 曲线通过一组控制点来定义. 我们先来看如何构造二阶 Bezier 曲线. 给定三个控制点 $\boldsymbol{P}_0, \boldsymbol{P}_1, \boldsymbol{P}_2$(这里的粗体表示原点到这一点的向量, 下同), 我们可以通过两轮线性插值得到曲线. 首先, 由参数 t 对 $\overrightarrow{P_0P_1}$ 和 $\overrightarrow{P_1P_2}$ 做线性插值:

$$\boldsymbol{Q}_0(t) = (1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1$$

$$\boldsymbol{Q}_1(t) = (1-t)\boldsymbol{P}_1 + t\boldsymbol{P}_2$$

然后用同一个参数 t 对 $\overrightarrow{Q_0Q_1}$ 做线性插值:

$$\boldsymbol{S}(t) = (1-t)\,\boldsymbol{Q}_0(t) + t\,\boldsymbol{Q}_1(t)$$

当 t 取遍 $[0,1]$ 时,$\boldsymbol{S}$ 对应的点的集合就是二阶 Bezier 曲线. 上述过程称作**德卡斯特里奥算法 (De Casteljau's Algorithm)**, 效率高, 编程方便, 因而被广泛使用.

类似地, n 阶 Bezier 曲线由 $n+1$ 个控制点 $\boldsymbol{P}_0, \boldsymbol{P}_1, \cdots, \boldsymbol{P}_n$ 通过 n 轮线性插值得到.

```python
def bezier(points: list[Vec2], t: float) -> Vec2:
    n = len(points)
    P = points.copy()
    for r in range(1, n):
        for i in range(n - r):
            P[i] = (1 - t) * P[i] + t * P[i + 1]
    return P[0]
```

下面推导 Bezier 曲线的显式表达式. 首先考虑 $n=2$ 的情形, 将前述表达式展开可得

$$\begin{aligned}\boldsymbol{S} &= (1-t)\,\boldsymbol{Q}_0 + t\,\boldsymbol{Q}_1 \\ &= (1-t)\left[(1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1\right] + t\left[(1-t)\boldsymbol{P}_1 + t\boldsymbol{P}_2\right] \\ &= (1-t)^2\boldsymbol{P}_0 + 2t(1-t)\boldsymbol{P}_1 + t^2\boldsymbol{P}_2\end{aligned}$$

可以发现,$\boldsymbol{P}_0$,$\boldsymbol{P}_1$ 和 $\boldsymbol{P}_2$ 的系数分别是 $((1-t)+t)^2$ 进行二项式展开的系数. 因此, 我们可以猜测[1]n 阶 Bezier 曲线的显式表达式为

$$\boldsymbol{S}(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^k t^{n-k} \boldsymbol{P}_k$$

其中

$$B_{n,k}(t) = \binom{n}{k} (1-t)^k t^{n-k}$$

被称作 n **阶伯恩斯坦多项式 (Bernstein Polynomial)**.

我们也可以用矩阵的形式来表示 Bezier 曲线. 例如, 三阶 Bezier 曲线可以通过以下的形式表示:

$$\boldsymbol{S}(t) = \boldsymbol{\tau}^{\mathrm{T}} \boldsymbol{M}_B \boldsymbol{u}$$

其中

$$\boldsymbol{\tau}^{\mathrm{T}} = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \qquad \boldsymbol{M}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \qquad \boldsymbol{u}^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{P}_0 & \boldsymbol{P}_1 & \boldsymbol{P}_2 & \boldsymbol{P}_3 \end{bmatrix}$$

1.2.2 Bezier 曲线的性质

首先有

$$\boldsymbol{S}(0) = \boldsymbol{P}_0 \qquad \boldsymbol{S}(1) = \boldsymbol{P}_n$$

因此 Bezier 曲线经过第一个和最后一个控制点.

其次有

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{S}(t)}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\sum_{k=0}^{n} \frac{n!}{k!(n-k)!}(1-t)^k t^{n-k}\boldsymbol{P}_k\right) \\ &= \sum_{k=0}^{n} \frac{n!}{k!(n-k)!}\left((n-k)(1-t)^k t^{n-k-1} - k(1-t)^{k-1}t^{n-k}\right)\boldsymbol{P}_k \\ &= n\sum_{k=0}^{n}\left(B_{n-1,k}(t) - B_{n-1,k-1}(t)\right)\boldsymbol{P}_k \\ &= n\sum_{k=0}^{n-1} B_{n-1,k}(t)(\boldsymbol{P}_{k+1} - \boldsymbol{P}_k)\end{aligned}$$

[1]事实上可以将这一过程与杨辉三角的构造相联系而证明.

于是

$$\boldsymbol{S}'(0) = n\left(\boldsymbol{P}_1 - \boldsymbol{P}_0\right) \qquad \boldsymbol{S}'(1) = n\left(\boldsymbol{P}_n - \boldsymbol{P}_{n-1}\right)$$

可见, Bezier 曲线在端点处的切线方向分别沿着 $\overrightarrow{P_0P_1}$ 和 $\overrightarrow{P_{n-1}P_n}$ 的方向.

定理 1.6 Bezier 曲线与端点的关系 设 Bezier 曲线的控制点依次为 $P_0, \cdots, P_n$, 则它经过 P_0 和 P_n, 且在端点处的切线方向分别沿着 $\overrightarrow{P_0P_1}$ 和 $\overrightarrow{P_{n-1}P_n}$ 的方向.

因此, 如果希望两条 Bezier 曲线首尾相接且切线连续, 只需让它们的端点重合且相邻的控制点共线即可.

于是, 三阶 Bezier 曲线是比较常用的, 既可以自由控制曲线的形状, 复杂度也不高.

Bezier 曲线的另一个重要性质是**凸包性质**.

定义 1.7 凸包性质 设 $\boldsymbol{P}_0, \boldsymbol{P}_1, \cdots, \boldsymbol{P}_n$ 为 Bezier 曲线的控制点, 那么 Bezier 曲线完全包含在由这些控制点构成的凸多边形内.

证明. 由于 $B_{n,k}(t) \geqslant 0$ 且 $\sum_{k=0}^{n} B_{n,k}(t) = 1$, 因此 $\boldsymbol{S}(t)$ 是控制点的凸组合, 从而在凸包内. □

1.2.3 Bezier 曲面

同样地, 我们可以用类似的方法绘制 Bezier 曲面. 这需要用到两个参数 u, v. 例如, 三阶 Bezier 曲面可以表示为

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_{3,i}(u)B_{3,j}(v)\boldsymbol{P}_{ij} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{M}_B\boldsymbol{P}\boldsymbol{M}_B^{\mathrm{T}}\boldsymbol{v}$$

其中

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{P}_{00} & \cdots & \boldsymbol{P}_{03} \\ \vdots & \ddots & \vdots \\ \boldsymbol{P}_{30} & \cdots & \boldsymbol{P}_{33} \end{bmatrix} \qquad \boldsymbol{v} = \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}$$

1.3 样条曲线

定义 1.8 样条曲线 样条曲线 (Spline Curve) 是计算机图形学和数值分析中常用的一类分段定义的多项式函数, 用于平滑地插值或逼近一组离散数据点.

给定点列 $\{\boldsymbol{P}_0, \cdots, \boldsymbol{P}_m\}$, 其中 $\boldsymbol{P}_i = (x_i, y_i)$. 样条曲线的确定实际上就是在每个区间 $[x_i, x_{i+1}]$ 上构造一个多项式 $S_i(x)$, 使其经过各点并保持一定的连续性. 常见的样条曲线有以下几种.

1.3.1 三次样条曲线

三次样条曲线是比较简单而常用的. 假定在每一段 $\{\boldsymbol{P}_i, \boldsymbol{P}_{i+1}\}$ 上定义参数 $t \in [0,1]$, 并且假定曲线上的点可以表示为

$$\boldsymbol{S}_i(t) = \boldsymbol{a}_i t^3 + \boldsymbol{b}_i t^2 + \boldsymbol{c}_i t + \boldsymbol{d}_i$$

曲线一共有 $4m$ 个未知数, 我们需要 $4m$ 个方程来确定这些未知数.

首先, 曲线需要经过各个点, 因此有

$$\boldsymbol{S}_i(0) = \boldsymbol{P}_i,\ \boldsymbol{S}_i(1) = \boldsymbol{P}_{i+1}$$

其次, 为了保持光滑性, 我们需要保证相邻段的切线方向一致, 因此有

$$\boldsymbol{S}_i'(1) = \boldsymbol{S}_{i+1}'(0)$$

最后, 为了保持曲线的平滑性, 我们还需要保证相邻段的二阶导数一致, 因此有

$$\boldsymbol{S}_i''(1) = \boldsymbol{S}_{i+1}''(0)$$

上述三个条件一共有 $4m-2$ 个方程. 为此, 我们还需要自行指定两个边界条件, 例如令曲线在端点处的二阶导数为 0, 称作**自然边界条件**:

$$\boldsymbol{S}_0''(0) = \boldsymbol{0},\ \boldsymbol{S}_{m-1}''(1) = \boldsymbol{0}$$

求解这一线性方程组即可得到三次样条曲线的参数.

从上述过程可以看出, 三次样条曲线是插值曲线, 并且具有 C^2 连续性. 然而, 改变任意一个控制点会影响整条曲线的形状, 因此三次样条曲线不具有局部性.

定义 1.9 局部性 如果改变一个控制点只会影响曲线的局部形状而不影响整体形状, 则称该插值曲线具有**局部性**.

1.3.2 Hermite 样条曲线

为了避免控制点对远端曲线形状的影响, 我们可以对三次样条曲线的构造做改进. 我们不要求各段曲线的二阶导连续, 而是指定各顶点 $\boldsymbol{P}_i$ 处的切线方向为 $\boldsymbol{p}_i$, 于是对于 $(\boldsymbol{P}_i, \boldsymbol{P}_{i+1})$ 之间的三次函数有

$$\boldsymbol{S}_i(0) = \boldsymbol{P}_i \qquad \boldsymbol{S}_i(1) = \boldsymbol{P}_{i+1} \qquad \boldsymbol{S}_i'(0) = \boldsymbol{p}_i \qquad \boldsymbol{S}_i'(1) = \boldsymbol{p}_{i+1}$$

这可以直接解得

$$\boldsymbol{S}_i(t) = \left(2t^3 - 3t^2 + 1\right)\boldsymbol{P}_i + \left(t^3 - 2t^2 + t\right)\boldsymbol{p}_i + \left(-2t^3 + 3t^2\right)\boldsymbol{P}_{i+1} + \left(t^3 - t^2\right)\boldsymbol{p}_{i+1}$$

这就是三次 Hermite 样条曲线. 其它阶数的 Hermite 样条曲线也可以类似地构造.

Hermite 样条曲线各控制点的导数方向可以自行指定, 也可以由控制点得到. 著名的 Catmull-Rom 样条曲

线就是一种特殊的 Hermite 样条曲线, 其切线方向由相邻控制点决定, 即

$$\boldsymbol{p}_i = \frac{\boldsymbol{P}_{i+1} - \boldsymbol{P}_{i-1}}{2}$$

相比三次样条曲线, Hermite 样条曲线同样是插值曲线, 但同时具有局部性. 相应地, 它在光滑程度上有所牺牲, 只有 C^1 连续性. 此外, 它并不需要解线性方程组, 结果已经一定, 计算效率更高[2].

1.3.3 B 样条曲线

与前面两种样条曲线相比,B 样条曲线不是插值曲线, 与 Hermite 样条曲线一样具有局部性, 但光滑性更好.

选定参数区间 $[a, b]$ 内一递增的数列 $a = t_0 < t_1 < \cdots < t_m = b$, 则 n 次 B 样条曲线可以表达为

$$\boldsymbol{b}(t) = \sum_{i=0}^{m} N_{i,n}(t)\boldsymbol{P}_i$$

其中 $N_{i,n}(t)$ 为 B 样条基函数, 通过递归定义:

$$N_{i,0}(t) = \begin{cases} 1, & t_i \leqslant t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

不难看出,$N_{i,p}(t)$ 是 p 阶的分段多项式, 并且在连接处保持了 C^{p-1} 连续性, 因此 n 阶 B 样条曲线具有 C^{n-1} 连续性. 此外,$N_{i,p}(t)$ 在 $[t_i, t_{i+p+1})$ 非 0, 因此改变 $\boldsymbol{P}_i$ 仅会影响 $[t_i, t_{i+p+1})$ 区间内的曲线形状. p 越大, 各个基函数和控制点影响的范围就越大, 局部性就越差.

B 样条曲线的一大优势是其灵活性. 我们可以分段选择不同阶数的基函数, 也可以选取重复节点 $t_i = t_{i+1} = \cdots = t_k$ 使得曲线其余部分连续性不变的情况下构造出尖锐的转折.

1.4 曲线光栅化

1.4.1 中点圆算法

中点圆算法 (Midpoint Circle Algorithm) 是 Bresenham 教授在 1962 年提出的一种高效的光栅化圆的算法, 其主要思路与 Bresenham 直线算法类似.

为了方便考虑, 我们假定要绘制的圆以原点为中心, 半径为 r. 由于圆的对称性, 我们只需计算第一象限的 1/8 圆, 然后将结果对称变换到其他象限即可.

考虑 $\frac{\pi}{4}$ 到 $\frac{\pi}{2}$ 的 1/8 圆, 我们从 $(0, r)$ 开始按顺时针方向绘制. 设当前绘制的点为 $P_i(x_i, y_i)$, 那么下一个点 P_{i+1} 也仅有两种可能: $P_{i+1} = (x_i + 1, y_i)$ 或 $P_{i+1} = (x_i + 1, y_i - 1)$. 我们用中点 $P_i' = \left(x_i + 1, y_i - \frac{1}{2}\right)$ 来判断下一个点的位置, 如果 P_i' 在圆内, 就向右移, 否则向右下移动. 圆的隐式方程为

$$F(x, y) = x^2 + y^2 - r^2 = 0$$

[2]Microsoft PowerPoint 提供的曲线工具就是三阶 Hermite 样条曲线, 用户可以自由编辑每个控制点的位置和对应的切线.

同样, 如果 $f\left(P_i'\right)<0$, 那么 P_i' 在圆内, 否则在圆外. 我们可以只对 $F\left(P_i'\right)$ 进行更新, 显然每次判断点的移动情况和像素的移动情况一样, 因此

$$F\left(P_{i+1}'\right)=F\left(x_{P_i'}+\Delta x, y_{P_i'}+\Delta y\right)=F\left(P_i'\right)+2x_{P_i'}\Delta x+(\Delta x)^2+2y_{P_i'}\Delta y+(\Delta y)^2$$

其中 $(\Delta x, \Delta y)=(1,0)$ 或 $(1,-1)$. 由于 P_i' 的坐标带有分数, 因此我们将上述式子转写为关于 P_i 的坐标的式子, 即

$$F\left(P_{i+1}'\right)=F\left(P_i'\right)+2x_i\Delta x+2\Delta x+(\Delta x)^2+2y_i\Delta y-\Delta y+(\Delta y)^2$$

于是向右更新像素 (即 $(\Delta x, \Delta y)=(1,0)$ 时) 需要将判断函数加上 $2x_i+3$, 向右下更新像素 (即 $(\Delta x, \Delta y)=(1,-1)$ 时) 需要将判断函数加上 $2x_i-2y_i+5$. 初始条件下有

$$F\left(1, r-\frac{1}{2}\right)=\frac{5}{4}-r$$

既然我们涉及的计算都是整数计算, 因此将初始值设为 $1-r$ 并不改变判断正负的结果. 当然, 如果半径 r 是浮点数, 就需要先计算上式后取整了.

综上, 我们可以将中点圆算法写成下面的程序:

```python
def draw_circle(xc: int, yc: int, r: int):
    x, y = 0, r
    F = 1 - r
    while x <= y:
        draw(xc + x, yc + y); draw(xc + y, yc + x)
        draw(xc - x, yc + y); draw(xc - y, yc + x)
        draw(xc + x, yc - y); draw(xc + y, yc - x)
        draw(xc - x, yc - y); draw(xc - y, yc - x)
        if F < 0: # F(P_i') < 0
            F += 2 * x + 3 # F(P_{i+1}) = F(P_i) + 2 * x_i + 3
        else: # F(P_i') >= 0
            y -= 1
            F += 2 * (x - y) + 5 # F(P_{i+1}) = F(P_i) + 2 * (x_i - y_i) + 5
        x += 1
```