Методи за рисуване на отсечки, окръжности и криви

1 Въведение

Разполагаме с матрица от цветове *pixel*, която е с размерите на екрана. В нея ще пишем RGB стойности, всяка компонента от който е реално число в [0, 1].

Да разгледаме *aliasing* проблема. Той най-лесно се вижда при рисуване на прави линии. Нека е дадена нормално уравнение на права:

$$\begin{aligned} (A, B) &\neq (0, 0) \\ A^2 + B^2 &= 1 \\ Ax + By + C &= 0 \end{aligned} \tag{1}$$

И дебелина r. Множеството от точките, който искаме да рисуваме са:

$$\|Ax + By + C\| \leq r \tag{2}$$

(2) разбива $\mathbb{R}^2$ на две непресичащи се множества. Тоест, всеки пиксел или рисуваме, или не. Дефинираме следния алгоритъм:

Algorithm 1 Aliased Line

```
function DrawLine(color)
    for all (x, y) do
        if Abs(A * x + B * y + C) ≤ r then
            pixel[x, y] ← color
```

На практика обаче работим с дискретен брой точки и такава стратегия изглежда лошо — правата е накъсана като трион. Освен това обхождаме всички пиксели, който е неоптимално.

1.1 Визуално подобрение

Въвеждаме два радиуса r и R, където $r < R$. Пикселите на разстояние по-малко от r оцветяваме напълно. Пикселите на разстояние повече от R игнорираме. Междинни стойности оцветяваме частично с интерполация на Ермит. Така правата постепенно ще се влива в фона.

```
function Mix(fg, bg t)
    return fg ∗ t + bg ∗ (1 − t)

function Smoothstep(xmin, xmax, x)
    if x ≤ xmin then
        return 0
    else if x ≥ xmax then
        return 1
    else
        t ← (x − xmin)/(xmax − xmin)
        return t ∗ t ∗ (3.0 − 2.0 ∗ t)
```

Използваме двете функции, за да реализираме:

Algorithm 2 Smooth Line

```
procedure DrawLine(color)
    for all (x, y) do
        distance ← Abs(A ∗ x + B ∗ y + C)
        t ← Smoothstep(r, R, distance)
        Mix(color, pixel[x, y], t)
```

1.2 Времево подобрение

Да се върнем към алгоритъма за чертаене на отсечка. For цикъла на алгоритъма Smooth Line може да обхожда всички точки в bounding box-а на отсечката. Проблемът е, че това е осезаемо бавно.

Ще направим по-хитро обхождане. Да фиксираме ред y_i от bbox-а. За него може да определим x_i, така че точката (x_i, y_i) да лежи на правата:

$$\begin{aligned} Ax_i + By_i + C = 0 \Leftrightarrow \\ Ax_i = -By_i - C \Leftrightarrow \\ x_i = (-By_i - C)/A \end{aligned}$$

Допускайки че $A \neq 0$. Ако $A = 0$, то правата е хоризонтална, т.е. лесно се рисува с друг алгоритъм. Сега търсим стъпка h>0 и $x_{max_i} = x_i + h$, така че $Ax_{max_i} + By_i + C = R$. По симетрия ще намерим и $x_{min_i} = x - h$:

$$\begin{aligned} A(x_i + h) + By_i + C = R \Leftrightarrow \\ (Ax_i + By_i + C) + Ah = R \Leftrightarrow \\ 0 + Ah = R \Leftrightarrow \\ h = -R/A \end{aligned}$$

Вече сме гарантирали, че $A \neq 0$. Стойността на h не зависи от i и ще я изчислим веднъж. Обхождайки колоните $[x_{min_i}, x_{max_i}]$ за всяко y_i, обхождаме оптимален брой пиксели.

2 Окръжности

Имаме всичко необходимо за чертаене на окръжности:

Algorithm 3 Drawing a Circle

```
1: procedure DRAW(center, r, color)
2:     for all (x, y) do
3:         distance ← ‖(x, y) − center‖
4:         t ← SMOOTHSTEP(r′, R′, distance)
5:         MIX(color, pixel[x, y], t)
```

Първи проблем е, че ще ни бъде подаден радиус r, от който да генерираме r' и R'. За целта използваме $r' = r$ и $R' = f(r')$. Практически $f(x) = x+2$ работи добре. При по-ниски стойности се губи ефекта, а при по-високи кръга става размазан.

Втори проблем е обхождането на ред 2. Да пробваме с всички точки от bbox-а на окръжността. Те са пропорционални на $4 * r^2$. А тези, които трябва да оцветим, са пропорционални на $\pi * r^2$. Следователно ще обходим $\dfrac{4 * r^2}{\pi * r^2} = 4/\pi$ пъти повече пиксели, което е в рамките на линейно.

3 Криви

Дадена е крива на bezier от степен n, дефинирана в интервала [0, 1]. Да приложим стратегия разделай и владей, като разбием интервала на две части — $[0, \frac{1}{2}] \cup [\frac{1}{2}, 1]$. Получаваме две криви, съответно C_1 и C_2, са от същата степен. Контролните им точки може да пресметнем с blossom функцията:

- C_1: $\mathbf{b}_i = \mathbf{b}(0^{<i>}, 0.5^{<n-i>})$
- C_2: $\mathbf{b}_i = \mathbf{b}(0.5^{<i>}, 1^{<n-i>})$

Сега трябва да решим същия проблем, но за по-малка крива. Продължаваме рекурсивно до определена дълбочина. За дъно на рекурсия може да апроксимираме кривата с отсечка между крайните й контролни точки. Да приложим алгоритъма за чертаене на права. Крайния резултат наподобява кривата, но има визуални артифакти. Проблемът се състои в комбинирането на отсечките. Най-добре се обяснява с картинка.

Краините точки съвпадат, но понеже отсечките не са успоредни, краищата не пасват. Този проблем присъства и при малки отсечки. Трябва ни друг алгоритъм.

Да се върнем към идеята за изчисляване на разстояние до геометричен обект. За дадена точка p търсим:

$$\min_{u \in [0,1]} \|\mathbf{b}(u) - p\| \tag{3}$$

Ще намерим приближена стойност на u, за който се достига минимума в (3). За целта ще решим същата задача, но за правата през крайните контролно точки.

$$u = \arg\min_{u \in [0,1]} \|u \times c_0 + (1-u)c_n - p\|$$

Ако имаме уравнението на правата през двете точки във вида (1), първо ще намерим ориентираното разстояние $s = Apx + Bpy + C$. Сега ще транслираме р с $-s\vec{n}$, където $\vec{n} = (A, B)$. Да проверим дали лежи на правата:

$$\begin{aligned}
A(p_x - s\vec{n_x}) + B(p_y - s\vec{n_x}) + C = 0 \Leftrightarrow \\
Ap_x - A^2 s + Bp_y - sB^2 + C = 0 \Leftrightarrow \\
(Ap_x + Bp_y + C) - s(A^2 + B^2) = 0 \Leftrightarrow \\
(Ap_x + Bp_y + C) = s
\end{aligned}$$

Предпоследният преход следва от факта, че $\|\vec{n}\| = 1$. Точката p_{proj} е проекцията на р върху правата. За да намерим u остава да сметнем коефицента на дължините между векторите $\overrightarrow{c_0 p_{proj}}$ и $\overrightarrow{c_0 c_n}$. Ето как ще стане без използва да използваме корен квадратен.

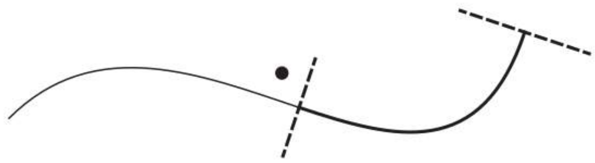
$$(\overrightarrow{c_0 p_{proj}} \cdot \overrightarrow{c_0 c_n}) = \|\overrightarrow{c_0 p_{proj}}\| \|\overrightarrow{c_0 c_n}\| \cos\theta$$

Понеже $\theta = 0$, то $\cos\theta = 1$. Сега ако разделим горното равенство с $\|\overrightarrow{c_0c_n}\|^2$, което е $(\overrightarrow{c_0c_n} \cdot \overrightarrow{c_0c_n})$, получаваме:

$$\frac{(\overrightarrow{c_0p_{proj}} \cdot \overrightarrow{c_0c_n})}{(\overrightarrow{c_0c_n} \cdot \overrightarrow{c_0c_n})} = \frac{\|\overrightarrow{c_0p_{proj}}\|}{\|\overrightarrow{c_0c_n}\|} = u$$

Сега предполагаме, че с намерението u, $\|\mathbf{b}(u) - p\|$ е много близо до стойността в (3). Изчисляваме $\mathbf{b}(u)$ чрез blossom функцията и с точката намираме разстоянието до p $\square$

Проблем представляват точките, който след проекция излизат извън отсечка. За тях е изпълнено $u \notin [0,1]$. Те са част от друго подразделение на кривата и трябва да бъдат отсечени. По-точно, правата през c_0 с направление, перпендикулярно на $b'(0)$, разделя равнината на две полупространства. Всички прави от 'грешното' полупространство трябва да бъдат игнорирани. Ето снимка от [1]:



В случая точката е в 'грешното' полупространство. Вижда се и защо е важно да изчисляваме производната.

Проверката я правим и за двата края на кривата. Ще решим проблема в единия, за другия е аналогично. Изчисляваме производната $\vec{b_0'} = c_1 - c_0$ с точност то линеен множител. Нека $\vec{u} = p - c_0$, $\theta = \sphericalangle(\vec{u}, \vec{b_0'})$. За θ трябва да е вярно:

$$-\frac{\pi}{2} + 2k\pi \leq \theta \leq \frac{\pi}{2} + 2k\pi \Leftrightarrow \cos\theta \geq 0 \Leftrightarrow (\vec{u} \cdot \vec{v}) \geq 0$$

Аналогично за края на отсечката.

3.1 Алгоритъм

Псевдокода е на следващта страница.

Ред 7 може да реализираме като вземем bbox-а на началната и крайна точка и го разширим с $\frac{r}{2}$ във четирите посоки. Не използваме всичките контролни точки за изчисляване на bbox-а от съображение за бързодействие.

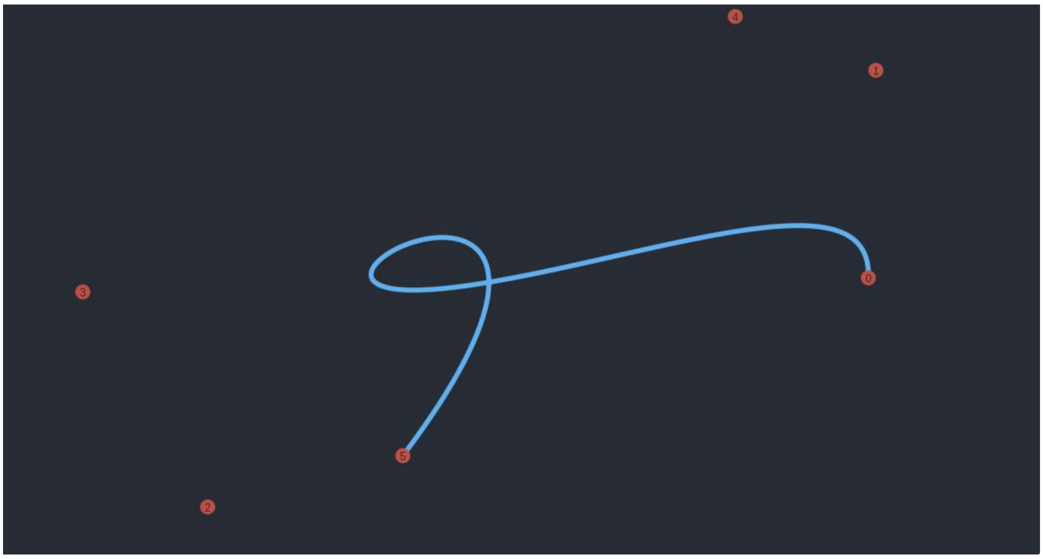
Функцията DrawCurve е важно да я викаме с малка крива. Иначе ще възпроизведе грешни резултати и цикъла на ред 7 ще отнеме твърде много време.

function ParametricOffset(a_0, a_1, p)
    $(A, B, C) \leftarrow$ normal line equation through a_0, a_1
    $d_{signed} \leftarrow Ap_x + Bp_y + C$
    $n_x \leftarrow A, n_y \leftarrow B$
    $p_{proj} \leftarrow p - s * \vec{n}$
    $\vec{\mathbf{a}} \leftarrow a_1 - a_0$
    $u \leftarrow (\overrightarrow{a_0 p_{proj}} \cdot \mathbf{a}) \;/\; \|\mathbf{a}\|^2$
    return u
function DrawCurve($c_0..c_n$, r, color)
    for all p $= (x, y)$ **do**
        $b'_0 \leftarrow c_1 - c_0$
        $b'_n \leftarrow c_n - c_{n-1}$
        if $b'_0 \cdot (p - c_0) < 0$ **then**
            skip this iteration
        if $b'_n \cdot (p - c_n) > 0$ **then**
            skip this iteration
        $u \leftarrow$ ParametricOffset(c_0, c_n, p)
        $p_{closest} \leftarrow$ Blossom$(c_0..c_n, u^{<n>})$
        $d \leftarrow \|p - p_{closest}\|$
        $t \leftarrow$ Smoothstep(r', R', d)
        Mix$(color, pixel[x][y], t)$

Algorithm 4 Drawing Curves

function Subdivide($c_0..c_n$)
    if recursion too deep **then**
        DrawCurve(c, r, color)
        return
    $C_{left} \leftarrow c_0^{left}..c_n^{left}$
    $C_{right} \leftarrow c_0^{right}..c_n^{right}$
    for i $\leftarrow$ 0 to n **do**
        $c_i^{left} \leftarrow$ Blossom$(c_0..c_n, 0^{<i>}, \frac{1}{2}^{<n-i>})$
        $c_i^{right} \leftarrow$ Blossom$(c_0..c_n, \frac{1}{2}^{<i>}, 1^{<n-i>})$
    Subdivide(C_{left})
    Subdivide(C_{right})

4 Результат



Литература

[1] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation*. Online edition.