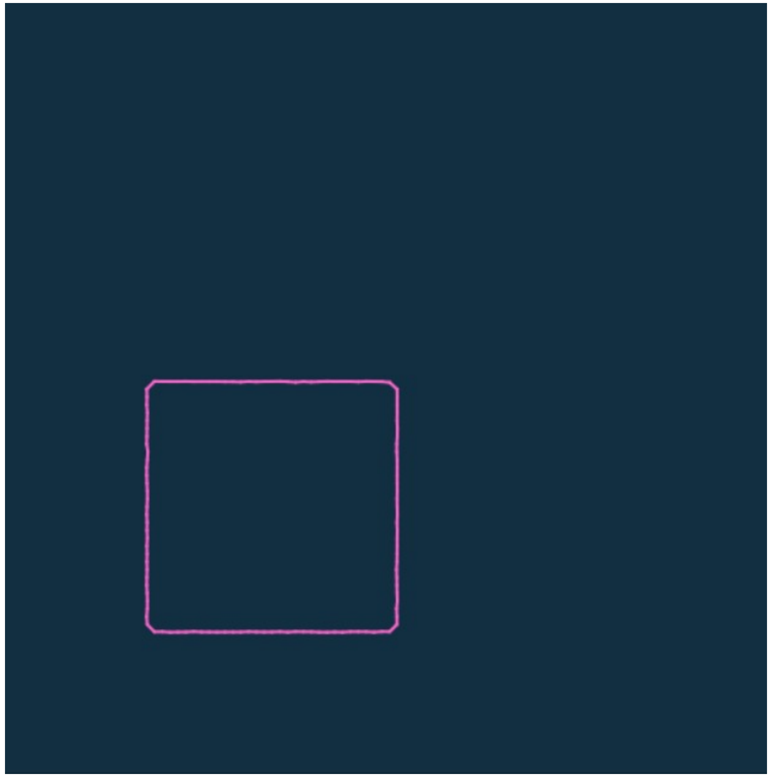
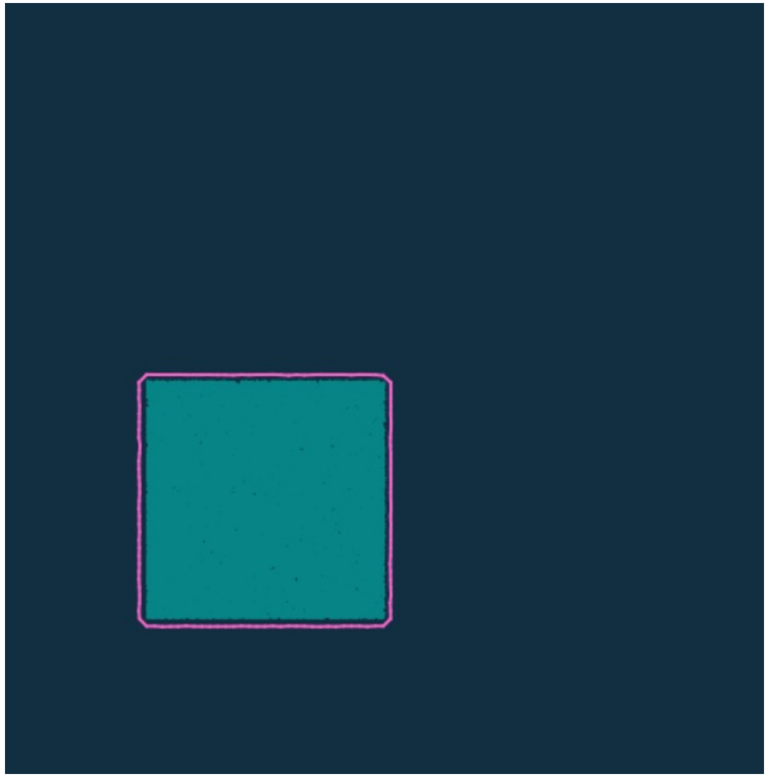
Marching Squares作业分享

wangfeng70117

2021.11.18





作业链接：
https://forum.taichi.graphics/t/1-mpm/1775
github：
https://github.com/wangfeng70117/surface_tension

隐式表面和显式表面

隐式表面就是不会告诉你任何点的信息，而是给出曲面上所有点的关系，我们使用2D举例，圆形的隐式曲面就是

$$x^2 + y^2 = r^2$$

一般我们会将隐式曲面的方程写为$f(x, y) = 0$,即$f(x, y) = x^2 + y^2 - r^2$

缺点：很难采样曲面上的点。

优点：很容易判断点与曲面的关系。

显式曲面

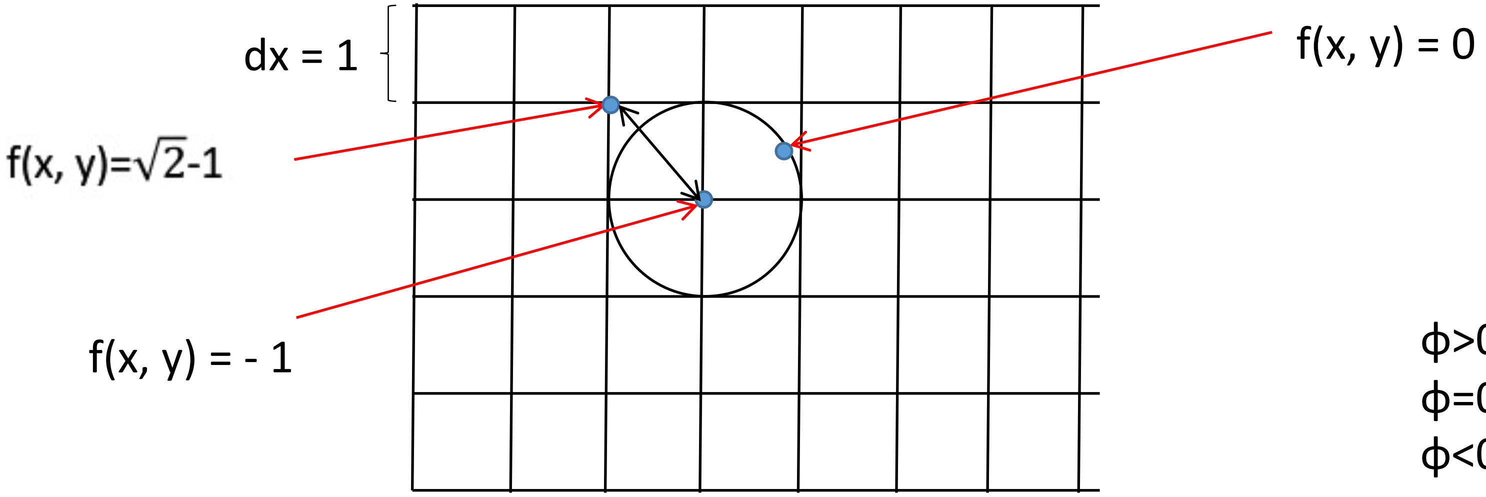
与隐式曲面相对应，显式曲面就是所有曲面上的点直接给出，或者通过某种映射关系直接得到。Marching Squares就是为了求出显式曲面。

优点：可以很容易的采样到所有的点

缺点：很难判断任意一点和表面的关系

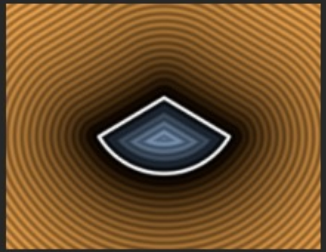
我的Marching Squares是先构建隐式曲面，再将隐式曲面转化为显式曲面得到的。

Sign Distance Field



$$\phi = f(x, y)$$

ϕ>0：点在圆的外部；
ϕ=0：点位于表面上；
ϕ<0：点在圆的内部。



Pie - exact (https://www.shadertoy.com/view/3l23RK)

```
float sdPie( in vec2 p, in vec2 c, in float r )
{
    p.x = abs(p.x);
    float l = length(p) - r;
    float m = length(p-c*clamp(dot(p,c),0.0,r)); // c=sin/cos of aperture
    return max(l,m*sign(c.y*p.x-c.x*p.y));
}
```

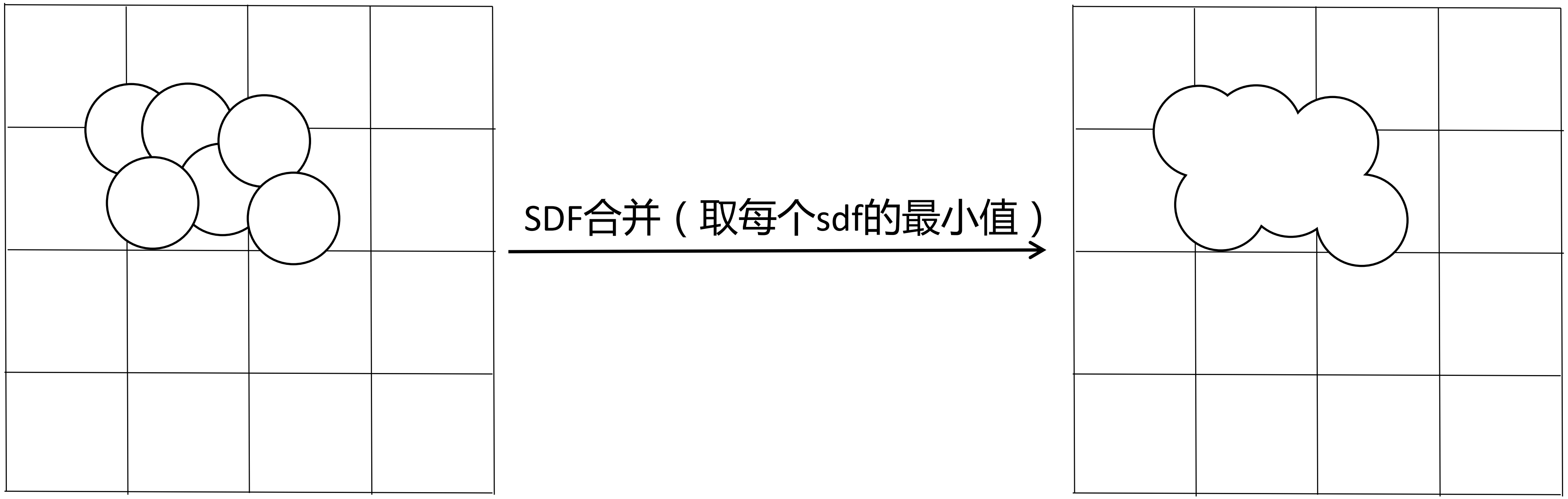
Arc - exact (https://www.shadertoy.com/view/wl23RK)

```
float sdArc( in vec2 p, in vec2 sca, in vec2 scb, in float ra, float rb )
{
    p *= mat2(sca.x,sca.y,-sca.y,sca.x);
    p.x = abs(p.x);
    float k = (scb.y*p.x>scb.x*p.y) ? dot(p,scb) : length(p);
    return sqrt( dot(p,p) + ra*ra - 2.0*ra*k ) - rb;
}
```

Horseshoe - exact (https://www.shadertoy.com/view/WlSGW1)

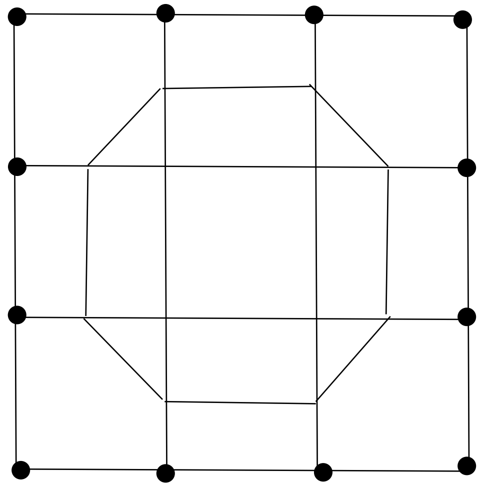
```
float sdHorseshoe( in vec2 p, in vec2 c, in float r, in vec2 w )
{
    p.x = abs(p.x);
    float l = length(p);
    p = mat2(-c.x, c.y, c.y, c.x)*p;
    p = vec2((p.y>0.0 || p.x>0.0)?p.x:l*sign(-c.x),
             (p.x>0.0)?p.y:l );
    p = vec2(p.x,abs(p.y-r))-w;
    return length(max(p,0.0)) + min(0.0,max(p.x,p.y));
```

https://www.iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm



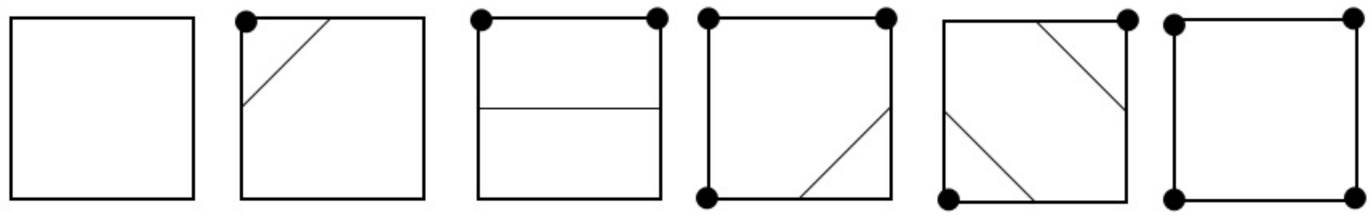

```python
# 生成level set隐式曲面
@ti.kernel
def gen_level_set():
    for i, j in ti.ndrange(diff_n_grid, diff_n_grid):
        min_dis = 10.0
        node_pos = ti.Vector([i * diff_dx, j * diff_dx])
        for I in range(particles.pos.shape[0]):
            distance = (particles.pos[I] - node_pos).norm() - radius
            if distance < min_dis:
                min_dis = distance
        sign_distance_field[i, j] = min_dis
```

Marching Squares



我们将离散的数据场中的每个体素单元作为一个网格，网格的每个顶点都携带对应的标量值，如果网格顶点上的标量值大于等值面的值则标记的“1”,如果小于等值面的值就标记为“0”。

在2D情况下，会有2^4 =16种情况，但是很多情况都可以通过旋转、翻转得到。

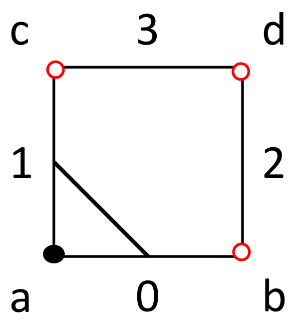


创建曲面上的边

```python
_et = np.array(
    [
        [[-1, -1], [-1, -1]],  #
        [[0, 1], [-1, -1]],  # a
        [[0, 2], [-1, -1]],  # b
        [[1, 2], [-1, -1]],  # ab
        [[1, 3], [-1, -1]],  # c
        [[0, 3], [-1, -1]],  # ca
        [[1, 3], [0, 2]],  # cb
        [[2, 3], [-1, -1]],  # cab
        [[2, 3], [-1, -1]],  # d
        [[2, 3], [0, 1]],  # da
        [[0, 3], [-1, -1]],  # db
        [[1, 3], [-1, -1]],  # dab
        [[1, 2], [-1, -1]],  # dc
        [[0, 2], [-1, -1]],  # dca
        [[0, 1], [-1, -1]],  # dcb
        [[-1, -1], [-1, -1]],  # dcab
    ],
    np.int32)
```

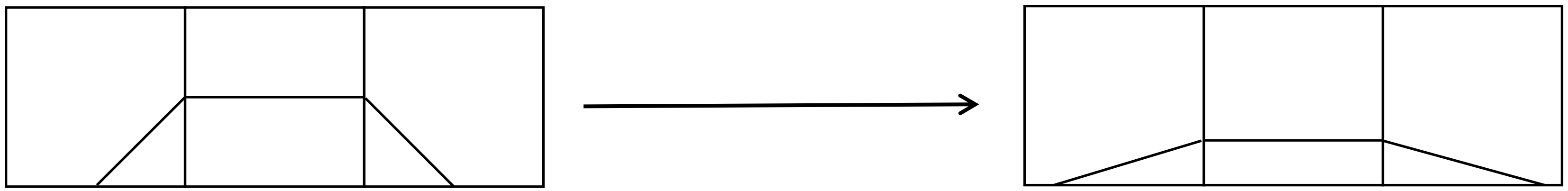
索引表：

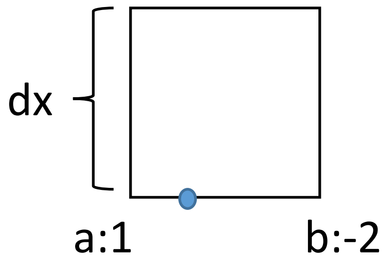
0	0	0	1




```python
# 将隐式曲面通过marching cube转化为显示曲面
@ti.kernel
def implicit_to_explicit():
    for i, j in ti.ndrange(diff_n_grid - 1, diff_n_grid - 1):
        id = 0
        if sign_distance_field[i, j] > 0: id |= 1
        if sign_distance_field[i + 1, j] > 0: id |= 2
        if sign_distance_field[i, j + 1] > 0: id |= 4
        if sign_distance_field[i + 1, j + 1] > 0: id |= 8
        for k in ti.static(range(2)):
            if et[id, k][0] != -1:
                n = ti.atomic_add(edge_num[None], 1)
                edge[n].begin_point = gen_edge_pos(i, j, et[id, k][0])
                edge[n].end_point = gen_edge_pos(i, j, et[id, k][1])
```

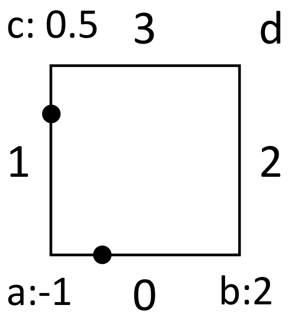
表面平滑





point.x = point(a).x + dx * abs(a) / (abs(a) + abs(b))
point.y = point(a).y

表面平滑



0: pos = [a.x + (abs(a) / (abs(a) + abs(b))), a.y]

1: pos = [a.x, a.y + (abs(a) / (abs(a) + abs(c)))]

```python
@ti.func
def gen_edge_pos(i, j, e):
    a = sign_distance_field[i, j]
    b = sign_distance_field[i + 1, j]
    c = sign_distance_field[i, j + 1]
    d = sign_distance_field[i + 1, j + 1]
    base_grid_pos = diff_dx * ti.Vector([i, j])
    result_pos = ti.Vector([.0, .0])
    if e == 0:
        result_pos = base_grid_pos + ti.Vector([(abs(a) / (abs(a) + abs(b))) * diff_dx, 0])
    if e == 1:
        result_pos = base_grid_pos + ti.Vector([0, (abs(a) / (abs(a) + abs(c))) * diff_dx])
    if e == 2:
        result_pos = base_grid_pos + ti.Vector([diff_dx, (abs(b) / (abs(b) + abs(d))) * diff_dx])
    if e == 3:
        result_pos = base_grid_pos + ti.Vector([(abs(c) / (abs(c) + abs(d))) * diff_dx, diff_dx])
    return result_pos
```

对3D Marching Cube有兴趣的朋友可以参考这个网站的教程：

http://paulbourke.net/geometry/polygonise/

我的流体仿真入门书籍：

《Fluid Engine Development》

《Fluid Simulation Computer Graphics》

我几乎没有产出的知乎：

https://www.zhihu.com/people/sui-yue-ru-ge-49-52-52