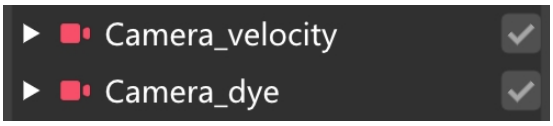
简单易懂的流体模拟实现

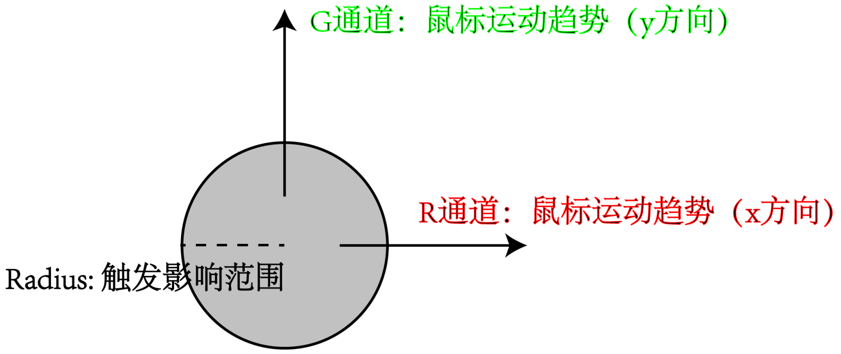
Ref: Webgl Fluid Simulation

1. 触发（包括速度场 + 颜色场）



▶ Camera_velocity ✓

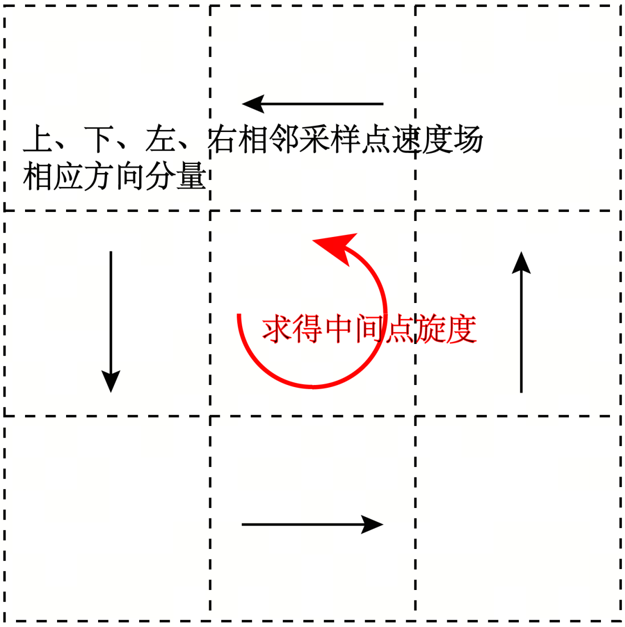
▶ Camera_dye ✓



```
vec3 splat = exp(-dot(p, p) / radius) * color;
//对于颜色场: color = 颜料颜色;
//对于速度场: color = 根据鼠标运动趋势换算为rg通道
gl_FragColor = vec4(base + splat, 1.0);
```

2. 根据相邻点速度，计算旋度

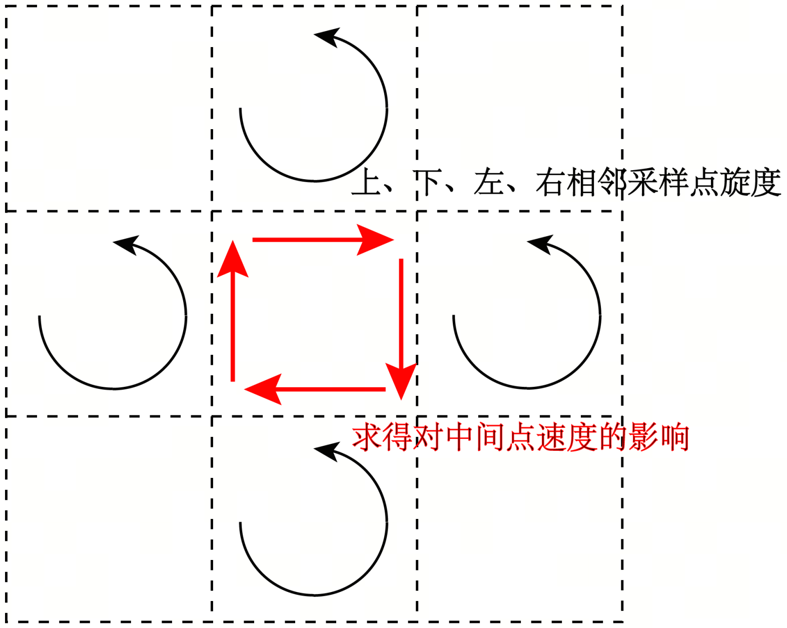
▶ Camera_curl



```
      float L = texture2D(uVelocity, vL).y;
      float R = texture2D(uVelocity, vR).y;
      float T = texture2D(uVelocity, vT).x;
      float B = texture2D(uVelocity, vB).x;
      float vorticity = R - L - T + B;
      gl_FragColor = vec4(0.5 * vorticity, 0.0, 0.0, 1.0);
      //逆时针为正，顺时针为负
```

3. 根据旋度计算涡量，再反映到速度

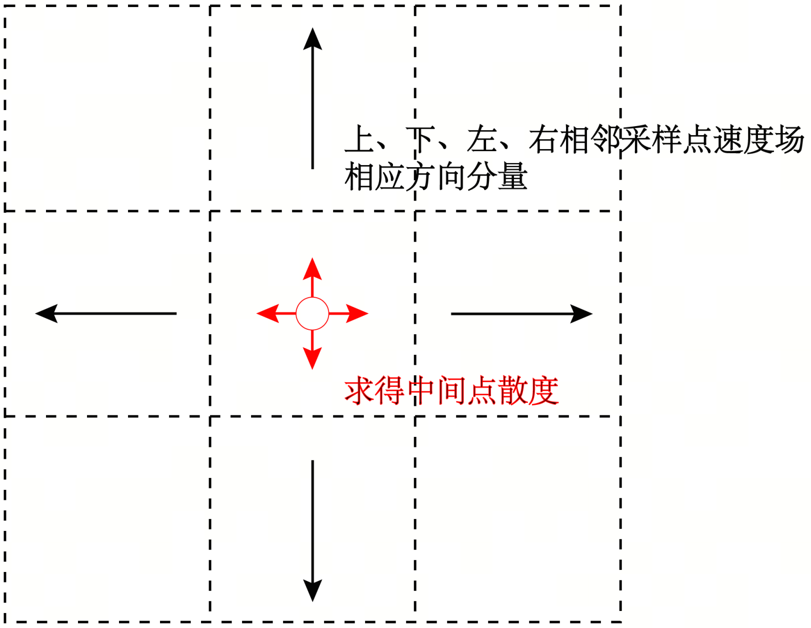
▶ Camera_vorticity



```
    vec2 force = 0.5 * vec2(abs(B) - abs(T), abs(R) - abs(L));
    force /= length(force) + 0.0001;
    force *= curl * C;

    vec2 velocity = texture2D(uVelocity, vUv).xy;
    velocity += force * dt;
```

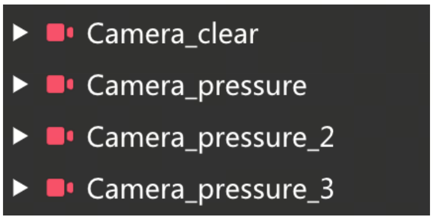
4. 根据相邻点速度，计算散度

▶ Camera_divergence



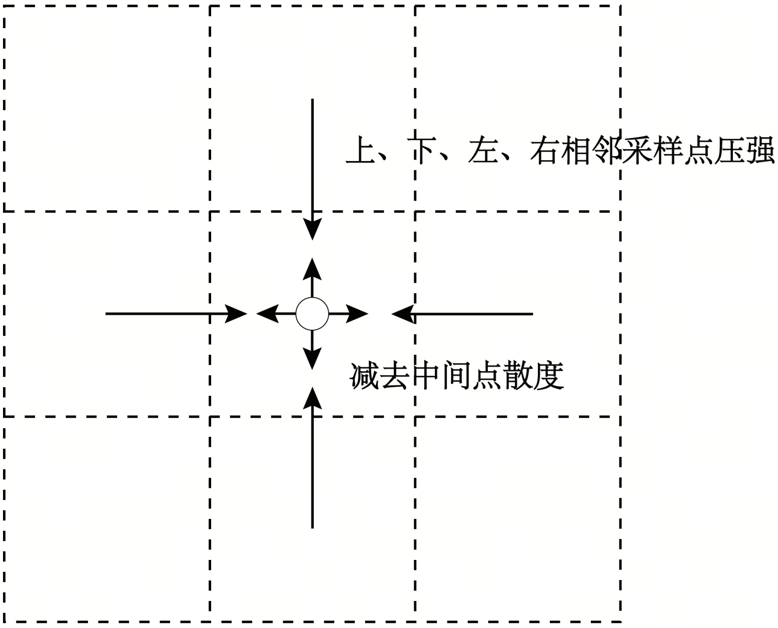
```
        float L = texture2D(uVelocity, vL).x;
        float R = texture2D(uVelocity, vR).x;
        float T = texture2D(uVelocity, vT).y;
        float B = texture2D(uVelocity, vB).y;

        vec2 C = texture2D(uVelocity, vUv).xy;
        if (vL.x < 0.0) { L = -C.x; }
        if (vR.x > 1.0) { R = -C.x; }
        if (vT.y > 1.0) { T = -C.y; }
        if (vB.y < 0.0) { B = -C.y; }//这一块是在屏幕边缘实现“触边反弹”

        float div = 0.5 * (R - L + T - B);
```

5. 相邻点压强与中间点散度抵消，计算中间点压强



▶ Camera_clear
▶ Camera_pressure
▶ Camera_pressure_2
▶ Camera_pressure_3

（这里的原理是：对于不可压缩流体，一旦散度不为零，即该点流入≠流出，即为不平衡状态；为了回到平衡状态，例如当散度>0即流出>流入时，为了保持该点平衡，该点压强会趋向于比周围压强小，目的是重新回到流入=流出状态）

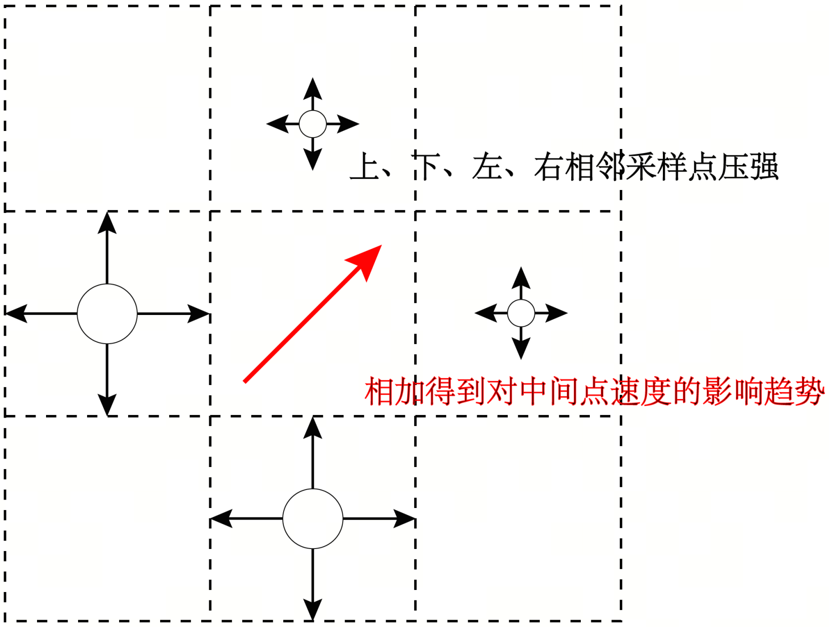


```
        float L = texture2D(uPressure, vL).x;
        float R = texture2D(uPressure, vR).x;
        float T = texture2D(uPressure, vT).x;
        float B = texture2D(uPressure, vB).x;
```

```
    float C = texture2D(uPressure, vUv).x;
    float divergence = texture2D(uDivergence, vUv).x;
    float pressure = (L + R + B + T - divergence) * 0.25;
    gl_FragColor = vec4(pressure, 0.0, 0.0, 1.0);
```

6. 相邻点压强再反映到速度场

▶ Camera_gradienSubtract

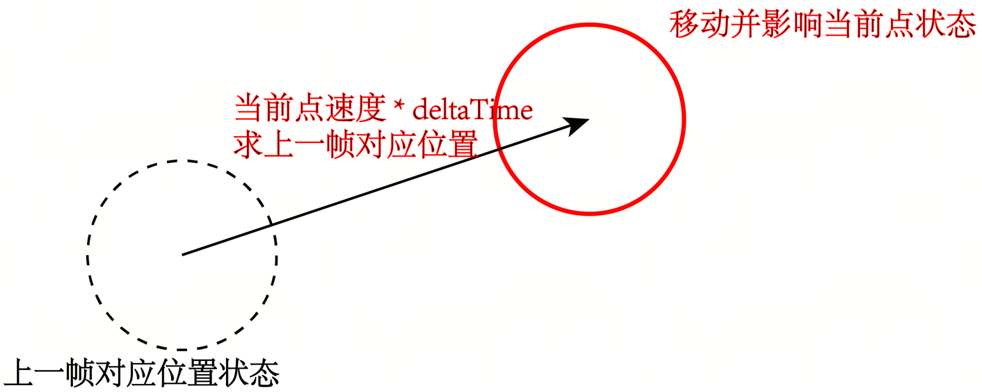


```
    float L = texture2D(uPressure, vL).x;
    float R = texture2D(uPressure, vR).x;
    float T = texture2D(uPressure, vT).x;
    float B = texture2D(uPressure, vB).x;
    vec2 velocity = texture2D(uVelocity, vUv).xy;
    velocity.xy -= vec2(R - L, T - B);
```

7. 用速度场计算对流

▶ Camera_advection

▶ Camera_advection_dye



```glsl
vec2 coord = vUv - dt * texture2D(uVelocity, vUv).xy * texelSize;
vec4 result = texture2D(uSource, coord);
// dt = deltatime
// 实际上是逆速度场变化趋势，取上一帧的场中对应方向相邻点状态，影响当前点
```

Reference

https://zhuanlan.zhihu.com/p/165479232

基于PBF的流体模拟

基于NS方程的流体模拟与应用

流体视效集合

欧拉视角：流体模拟从示例代码开始

Bonus

如果改变采样单位，减小速度系数等，还能获得史莱姆效果