```cpp
#include "sphfluidsystem.h"
#include "PhysicsUtilities.h"
#include "kernelutilities.h"

using namespace std;

SPHFluidSystem::SPHFluidSystem()
{
    SPHFluidSystem(5000);
}

SPHFluidSystem::SPHFluidSystem(int numParticles) : ParticleSystem(numParticles)
{
    initConstants();
    //buildTestSystem2();
    testOneInitializeSystem();
    //buildTestSystem2();
    Vector3f origin = Vector3f::ZERO;
    particleGrid = ParticleGrid(origin, 0.5 , 0.9, 0.5);
    vecParticleDensities = vector<float>();
    vecParticlePressures = vector<float>();
}

void SPHFluidSystem::initConstants()
{
    PARTICLE_MASS = 0.02;
    GRAVITY_CONSTANT = 6.5;
    REST_DENSITY = 1000.0;
    GAS_CONSTANT = 1.0;
    VISCOSITY_CONSTANT = 6.0;
    TENSION_CONSTANT = 0.10;
    TENSION_THRESHOLD = 6.0;
    SELF_DENSITY_CONSTANT = PARTICLE_MASS *
KernelUtilities::polySixKernel(Vector3f::ZERO);
    SELF_LAPLACIAN_COLOR_FIELD = PARTICLE_MASS *
KernelUtilities::laplacianPolySixKernel(Vector3f::ZERO);
}

SPHFluidSystem::~SPHFluidSystem()
{

}

vector<Vector3f> SPHFluidSystem::evalF(vector<Vector3f> state)
{
    vector<Vector3f> particlePositionsInState =
PhysicsUtilities::getParticlePositions(state);
    particleGrid.initializeGrid(particlePositionsInState);
    calculateDensitiesAndPressures(state);

    vector<Vector3f> derivative;
    for (int particleIndex = 0; particleIndex < m_numParticles; ++particleIndex)
    {
        Vector3f positionOfParticle = PhysicsUtilities::getPositionOfParticle(state,
particleIndex);
        Vector3f velocityOfParticle = PhysicsUtilities::getVelocityOfParticle(state,
particleIndex);
        float densityAtParticleLoc = vecParticleDensities[particleIndex];
        float pressureAtParticleLoc = vecParticlePressures[particleIndex];


        // CALCULATE PRESSURE AND VISCOSITY FORCES
        // ALSO CALCULATE PIECES OF THE SURFACE TENSION FORCE
        Vector3f totalPressureForce = Vector3f::ZERO;
        Vector3f totalViscosityForce = Vector3f::ZERO;
        Vector3f totalGradColorField = Vector3f::ZERO;
        float totalLaplacianColorField = 0.0f;

        vector<int> neighborIndexes =
```

```
particleGrid.getNeighborParticleIndexes(particleIndex, positionOfParticle);
        for (int neighborI : neighborIndexes)
        {
            Vector3f positionOfNeighbor = PhysicsUtilities::getPositionOfParticle(state,
neighborI);

            float densityAtNeighborLoc = vecParticleDensities[neighborI];
            float pressureAtNeighborLoc = vecParticlePressures[neighborI];

            Vector3f spikyKernelGradForDebugging;

            Vector3f pressureContribution;
            Vector3f viscosityContribution;
            Vector3f gradColorFieldContribution;
            float laplacianColorFieldContribution;

            Vector3f rForKernel = positionOfParticle - positionOfNeighbor;

            float rEpsilon = 0.0005;
            if (rForKernel.abs() < rEpsilon)
            {

                pressureContribution = Vector3f::ZERO;
                viscosityContribution = Vector3f::ZERO;
                gradColorFieldContribution = Vector3f::ZERO;
                laplacianColorFieldContribution = 0.0f;

                //rForKernel = Vector3f(0.003, 0.003, 0.003);
            }

            else
            {
                // CALCULATE PRESSURE CONTRIBUTION FROM THE NEIGHBOR
                Vector3f spikyKernelGrad = KernelUtilities::gradSpikyKernel(rForKernel);
                spikyKernelGradForDebugging = spikyKernelGrad;

                if (isNan(spikyKernelGrad))
                {
                    cout << "NAN spiky kernel gradient" << endl;
                    cout << "Spiky kernel gradient: ";
DebugUtilities::printVector3f(spikyKernelGrad);
                    cout << "ri - rj magnitude: " << rForKernel.abs() << endl;
                }

                pressureContribution = PhysicsUtilities::getPressureForce(
PARTICLE_MASS,
                                                                          pressureAtPar
ticleLoc,
                                                                          pressureAtNei
ghborLoc,
                                                                          densityAtNeig
hborLoc,
                                                                          spikyKernelGr
ad );

                // CALCULATE VISCOSITY CONTRIBUTION FROM THE NEIGHBOR
                float laplacianKernel =
KernelUtilities::laplacianViscosityKernel(rForKernel);
                Vector3f velocityOfNeighbor =
PhysicsUtilities::getVelocityOfParticle(state, neighborI);
                viscosityContribution =
PhysicsUtilities::getViscosityForce(PARTICLE_MASS,
                                                                           VISCOSITY_CO
NSTANT,
                                                                           densityAtNei
ghborLoc,
                                                                           laplacianKer
nel,
                                                                           velocityOfNe
```

```
ighbor,
                                                                velocityOfPa
rticle);

                // CALCULATE GRAD COLOR FIELD CONTRIBUTION FROM NEIGHBOR
                Vector3f gradPolySixKernel =
KernelUtilities::gradPolySixKernel(rForKernel);
                gradColorFieldContribution = (PARTICLE_MASS / densityAtNeighborLoc) *
gradPolySixKernel; // Eq(15);

                // CALCULATE LAPLACIAN COLOR FIELD CONTRIBUTION FROM NEIGHBOR
                float laplacianPolySixKernel =
KernelUtilities::laplacianPolySixKernel(rForKernel);
                laplacianColorFieldContribution = (PARTICLE_MASS / densityAtNeighborLoc)
* laplacianPolySixKernel; // Eq(15)

                totalViscosityForce += viscosityContribution;
                totalPressureForce += pressureContribution;
                totalGradColorField += gradColorFieldContribution;

                //cout << "Grad color field contribution: ";
DebugUtilities::printVector3f(gradColorFieldContribution);

                // For debugging
                if(isNan(viscosityContribution))
                {
                    cout << "Encountered NAN viscosity contribution" << endl;
                    cout << "Viscosity contribution: ";
DebugUtilities::printVector3f(viscosityContribution);

                    cout << "Particle loc: ";
DebugUtilities::printVector3f(positionOfParticle);
                    cout << "Particle velocity: ";
DebugUtilities::printVector3f(velocityOfParticle);

                    cout << "NeighborI: " << neighborI << endl;
                    cout << "Neighbor loc: ";
DebugUtilities::printVector3f(positionOfNeighbor);
                    Vector3f velocityNeighbor =
PhysicsUtilities::getVelocityOfParticle(state, neighborI);
                    cout << "Neighbor velocity: ";
DebugUtilities::printVector3f(velocityNeighbor);
                    cout << "Density at neighbor loc: " << densityAtNeighborLoc << endl;

                    float laplacianKernel =
KernelUtilities::laplacianViscosityKernel(positionOfParticle - positionOfNeighbor);
                    cout << "Laplacian Kernel: " << laplacianKernel << endl;

                    assert(false); // Kill execution of the program
                }

                if(isNan(pressureContribution))
                {
                    cout << "Encountered NAN pressure contribution" << endl;
                    cout << "Pressure contribution: ";
DebugUtilities::printVector3f(pressureContribution);

                    cout << "Particle loc: ";
DebugUtilities::printVector3f(positionOfParticle);
                    cout << "Pressure at particle loc: " << pressureAtParticleLoc <<
endl;

                    cout << "NeighborI: " << neighborI << endl;
                    cout << "Neighbor loc: ";
DebugUtilities::printVector3f(positionOfNeighbor);
                    cout << "Density at neighbor loc: " << densityAtNeighborLoc << endl;
                    float pressureAtNeighborLoc =
PhysicsUtilities::getPressureAtLocation(densityAtNeighborLoc, REST_DENSITY,
GAS_CONSTANT);
```

```
                    cout << "Pressure at neighbor loc: " << pressureAtNeighborLoc <<
endl;

                    cout << "Spiky kernel grad: ";
DebugUtilities::printVector3f(spikyKernelGradForDebugging);

                    assert(false); // Kill execution of the program
                }

                if (isNan(gradColorFieldContribution))
                {
                    cout << "Encountered NAN grad color field contribution" << endl;
                    cout << "Grad color field contribution: ";
DebugUtilities::printVector3f(gradColorFieldContribution);
                    assert(false);
                }

                if (isNan(laplacianColorFieldContribution))
                {
                    cout << "Encountered NAN laplacian color field contribution" <<
endl;
                    cout << "Laplacian color field contribution: " <<
laplacianColorFieldContribution << endl;
                    assert(false);
                }
            }

            totalViscosityForce += viscosityContribution;
            totalPressureForce += pressureContribution;
            totalGradColorField += gradColorFieldContribution;
            totalLaplacianColorField += laplacianColorFieldContribution;

        }

        totalLaplacianColorField += (SELF_LAPLACIAN_COLOR_FIELD / densityAtParticleLoc);

        // CALCULATE TOTAL SURFACE TENSION FORCE FROM COMPONENTS
        Vector3f surfaceNormal = totalGradColorField;
        float surfaceNormalMag = surfaceNormal.abs();

        Vector3f totalSurfaceTensionForce;

        if (surfaceNormalMag > TENSION_THRESHOLD)
        {
            float constant = (-1.0 * TENSION_CONSTANT * totalLaplacianColorField) /
surfaceNormalMag;
            totalSurfaceTensionForce = constant * surfaceNormal;
        }

        else
        {
            //cout << "Surface normal mag: " << surfaceNormalMag << endl;
            totalSurfaceTensionForce = Vector3f::ZERO;
        }

        if (totalSurfaceTensionForce.abs() > 0.0f)
        {
            //cout << "Total surf tension force: ";
DebugUtilities::printVector3f(totalSurfaceTensionForce);
            //cout << "Total grad color field: ";
DebugUtilities::printVector3f(totalGradColorField);
            //cout << "Total laplacian color field: " << totalLaplacianColorField <<
endl;
        }
        // COMPUTE TOTAL ACCELERATION OF PARTICLE
        Vector3f accelPressure = totalPressureForce / densityAtParticleLoc;
        Vector3f accelViscosity = totalViscosityForce / densityAtParticleLoc;
        Vector3f accelSurfaceTension = totalSurfaceTensionForce / densityAtParticleLoc;
        Vector3f accelGravity = PhysicsUtilities::getGravityForce(PARTICLE_MASS,
```

```
GRAVITY_CONSTANT) / PARTICLE_MASS;

        Vector3f accelTotal = accelPressure + accelViscosity + accelGravity +
accelSurfaceTension;

        derivative.push_back(velocityOfParticle);
        derivative.push_back(accelTotal);
    }

    return derivative;
}

void SPHFluidSystem::draw()
{
    int numNanPositions = 0;
    for (int i = 0; i < m_numParticles; i++)
    {
        // Draw the particles
        Vector3f posParticle = PhysicsUtilities::getPositionOfParticle(m_vVecState, i);
        if (isNan(posParticle))
        {
            cout << "Encountered NAN position: " << "( " << posParticle.x() << " , " <<
posParticle.y() << " ," << posParticle.z() << " )" << endl;
            //++numNanPositions;
        }
        glPushMatrix();
        glTranslatef(posParticle[0], posParticle[1], posParticle[2] );
        glutSolidSphere(0.015f, 10.0f,10.0f);
        glPopMatrix();
    }

    //cout << "Num nans: " << numNanPositions << endl;
}

void SPHFluidSystem::reinitializeSystem()
{

}

// Helper functions
void SPHFluidSystem::calculateDensitiesAndPressures(vector<Vector3f> &state)
{
    vecParticleDensities = vector<float>();
    vecParticlePressures = vector<float>();

    for (int i = 0; i < m_numParticles; ++i)
    {
        float density = 0;
        Vector3f pos = PhysicsUtilities::getPositionOfParticle(state, i);
        vector<int> neighborIndexes = particleGrid.getNeighborParticleIndexes(i, pos);

        for (int neighborI : neighborIndexes)
        {
            Vector3f neighborPos = PhysicsUtilities::getPositionOfParticle(state,
neighborI);
            density += PARTICLE_MASS * KernelUtilities::polySixKernel(pos -
neighborPos);
        }

        density += SELF_DENSITY_CONSTANT;
        vecParticleDensities.push_back(density);

        float pressure = PhysicsUtilities::getPressureAtLocation(density, REST_DENSITY,
GAS_CONSTANT);
        vecParticlePressures.push_back(pressure);
    }
}

bool SPHFluidSystem::isNan(float val)
```

```
{
    return val != val;
}

bool SPHFluidSystem::isNan(Vector3f vec)
{
    return isNan(vec.x()) || isNan(vec.y()) || isNan(vec.z());
}

// Different system initializtions
void SPHFluidSystem::buildTwoParticleSystemNotNeighbors()
{
    Vector3f pos1(0.3, 0.5, 0.1);
    Vector3f pos2(0.1, 0.5, 0.1);

    m_vVecState.push_back(pos1);
    m_vVecState.push_back(Vector3f::ZERO);
    m_vVecState.push_back(pos2);
    m_vVecState.push_back(Vector3f::ZERO);

    m_numParticles = 2;
}

void SPHFluidSystem::buildTwoParticleSystemNeighbors()
{
    Vector3f pos1(0.3, 0.5, 0.1);
    Vector3f pos2(0.305, 0.5, 0.1);

    m_vVecState.push_back(pos1);
    m_vVecState.push_back(Vector3f::ZERO);
    m_vVecState.push_back(pos2);
    m_vVecState.push_back(Vector3f::ZERO);

    m_numParticles = 2;
}

void SPHFluidSystem::testOneInitializeSystem()
{
    for (int k = 0; k < 20; k++)
    {
        for (int i = 0; i < 15; i++ )
        {
            for (int j = 0; j < 20; j++) {
                Vector3f point(0.14 + .01 * i + .005 * (i %2), 0.24 + j * .01 +  .005 *
(j %2), 0.1 +  k * .01 + .005 * (k %2));
                m_vVecState.push_back(point);
                m_vVecState.push_back(Vector3f::ZERO);
            }
        }
    }

    m_numParticles = 6000;
}

void SPHFluidSystem::build2DTestSystem()
{
    float k = 0.4;
    for (int i = 0; i < 10; ++i)
    {
        for (int j = 0; j < 10; ++j)
        {
            Vector3f point(0.04 * (i + 1), 0.04 * (j + 1), k);
            m_vVecState.push_back(point);
            m_vVecState.push_back(Vector3f::ZERO);

        }
    }

    m_numParticles = 100;
```

```
}

void SPHFluidSystem::buildTestSystem2()
{
    for (float x = 0.1; x <= 0.384; x += 0.02)
    {
        for (float y = 0.1; y <= 0.576; y += 0.02)
        {
            for (float z = 0.1; z <= 0.384; z+= 0.02)
             {
                Vector3f point(x, y, z);
                m_vVecState.push_back(point);
                m_vVecState.push_back(Vector3f::ZERO);
             }
        }
    }

    m_numParticles = m_vVecState.size() / 2;
}
```