SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

THE PREMIER CONFERENCE & EXHIBITION ON
COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

FLUID EXAMPLE IN OPENVDB

ANDRE PRADHANA, NVIDIA

GOVERNING EQUATIONS

NAVIER-STOKES EQUATION

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = \rho\mathbf{g} - \nabla p$$

$$\nabla\cdot\mathbf{v} = 0$$

SPLIT

1. Body Forces, i.e. apply gravity

$$\frac{\partial \mathbf{v}}{\partial t} = \mathbf{g}$$

2. Enforce incompressibility/ remove divergence

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho}\nabla p = 0, \text{ such that } \nabla \cdot \mathbf{v} = 0.$$

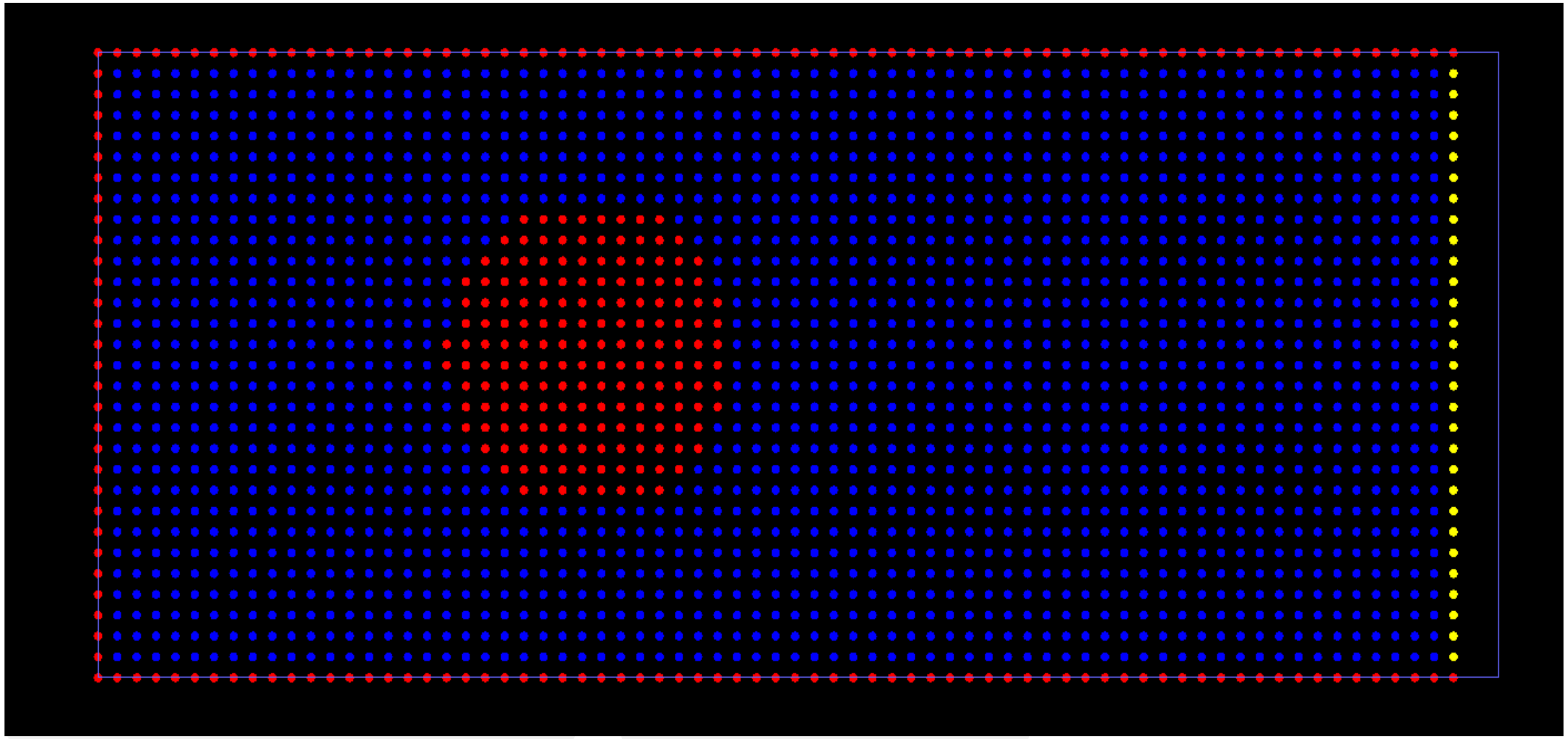
3. Advection, i.e. move velocity field (and density field)

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = 0$$

POISSON EQUATION

$$\begin{aligned} \nabla^2 p(\mathbf{x}) &= \nabla \cdot \mathbf{v}(\mathbf{x}), \quad && \mathbf{x} \in \Omega \\ p(\mathbf{x}) &= f_D(\mathbf{x}) && \partial\Omega_D \\ \frac{\partial p}{\partial \mathbf{n}}(\mathbf{x}) &= f_N(\mathbf{x}) && \partial\Omega_N \end{aligned}$$

TYPES OF DOFS IN A FLUID SIM



- Red: Collider/Neumann pressure DOF
- BLUE: Fluids/Pressure DOF
- Yellow: Free surface, opening/Dirichlet DOF

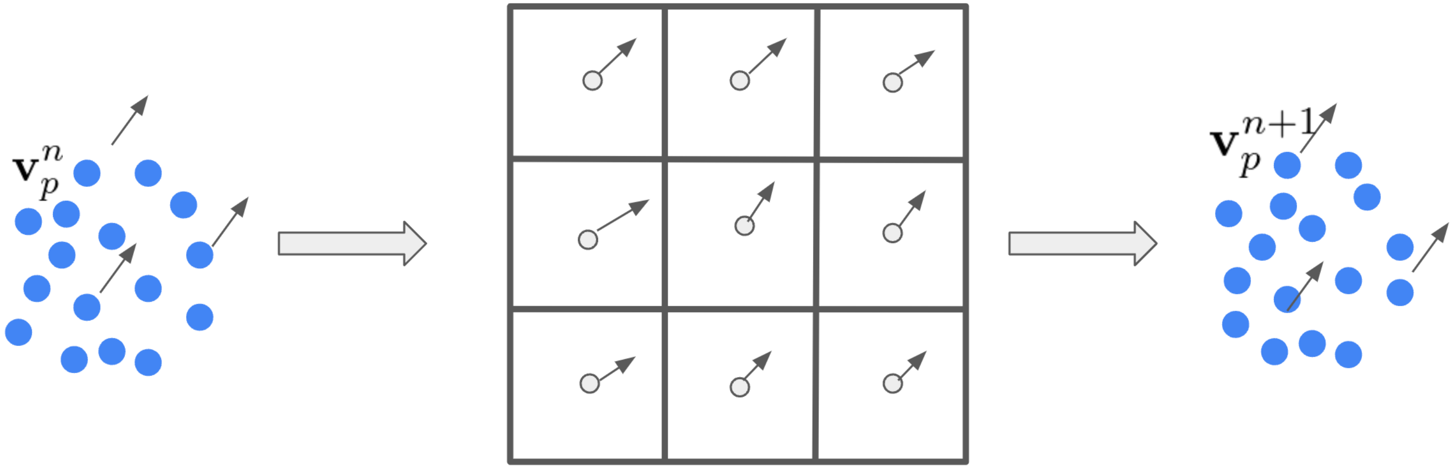
THE PREMIER CONFERENCE & EXHIBITION ON
COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

FLIP SOLVER

WHAT IS A FLIP SIMULATION

1. Particles to Grid
2. Grid Velocity Update
   a. Apply gravity
   b. Remove divergence
3. Grid to Particles
4. Particles Advection



INITIALIZATION

- Particles states: stored as `PointDataGrid`

```
points::PointDataGrid::Ptr mPoints;
```

- Voxel grids:

```
FloatGrid::Ptr mCollider;

BoolGrid::Ptr mInteriorPressure;
Vec3SGrid::Ptr mVCurr;
Vec3SGrid::Ptr mVNext;
Vec3SGrid::Ptr mVDiff; // For FLIP (Fluid Implicit Particle)

FloatGrid::Ptr mPressure;
FloatGrid::Ptr mDivBefore;
FloatGrid::Ptr mDivAfter;
```

INITIALIZE SIMULATION STATES

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

- Initialize particles by sampling it in a VDB, and append particle states

```
mPoints = points::denseUniformPointScatter(*fluidLSInit, mPointsPerVoxel);
mPoints->setName("Points");
points::appendAttribute<Vec3s>(mPoints->tree(),
    "velocity" /* attribute name */,
    Vec3s(0.f, 0.f, 0.f) /* uniform value */,
    1 /* stride or total count */,
    true /* constant stride */,
    nullptr /* default value */,
    false /* hidden */,
    false /* transient */);
```

- Set up collider
- Create interior mask for interior pressure DOF

```
mInteriorPressure = BoolGrid::create(false);
mInteriorPressure->tree().topologyUnion(mPoints->tree());
mInteriorPressure->tree().topologyDifference(mCollider->tree());
mInteriorPressure->tree().voxelizeActiveTiles();
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

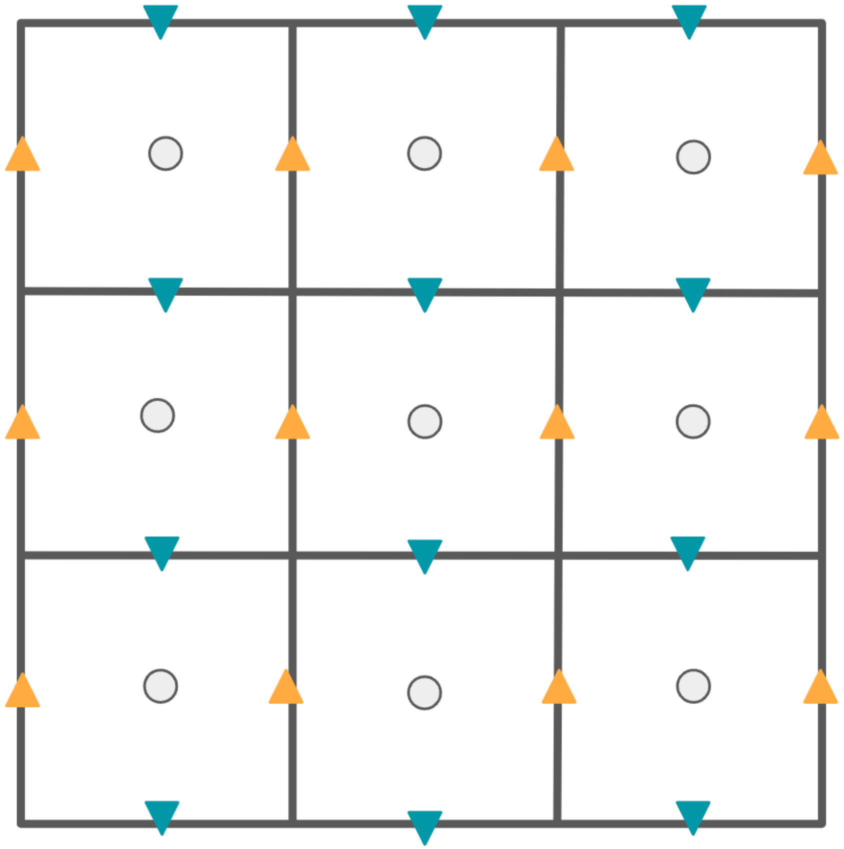
FLIP::PARTICLES TO GRID

PARTICLES TO GRID

- Trilinear interpolation to (staggered) grid: $\mathbf{v}_i = \sum_p \mathbf{v}_p N_i(\mathbf{x}_p)$
- Rasterize particle velocity to grid

```
void
FlipSolver::particlesToGrid(){
    TreeBase::Ptr baseVTree = points::rasterizeTrilinear<true /* staggered */, Vec3s>(mPoints-  >tree(),
"velocity");

    Vec3STree::Ptr velTree = DynamicPtrCast<Vec3STree>(baseVTree);
    mVCurr = Vec3SGrid::create(velTree);
    mVCurr->setGridClass(GRID_STAGGERED);
    mVCurr->setTransform(mXform);
    mVCurr->setName("v_curr");

    mVNext = mVCurr->deepCopy();
    mVNext->setName("v_next");

    // For FLIP update
    mVDiff = mVCurr->deepCopy();
    mVDiff->setName("v_flip");
}
```

STAGGERED/MAC GRID



THE PREMIER CONFERENCE & EXHIBITION ON
COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

FLIP::GRID VELOCITY UPDATE

FLIP::GRID VELOCITY UPDATE

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

- Add body forces: $\frac{\partial \mathbf{v}}{\partial t} = \mathbf{g}$

- Implementation

```
struct ApplyGravityOp
{
ApplyGravityOp(float const dt, Vec3s const gravity) : dt(dt), gravity(gravity) {}

template <typename T>
void operator()(T &leaf, size_t) const {
    for (typename T::ValueOnIter iter = leaf.beginValueOn(); iter; ++iter) {
        Vec3s newVal = *iter + dt * gravity;
        iter.setValue(newVal);
    }
}

Vec3s const gravity;
float const dt;
};// ApplyGravityOp
```

FLIP::APPLY GRAVITY

- Calls it with `Leaf Manager foreach`

```
void
FlipSolver::addGravity(float const dt) {
    tree::LeafManager<Vec3STree> lm(mVCurr->tree()); // LeafManager.h
    FlipSolver::ApplyGravityOp op(dt, mGravity);
    lm.foreach(op);
}
```

```
struct ApplyGravityOp
{
   . . .
template <typename T>
void operator()(T &leaf, size_t) const {
    for (typename T::ValueOnIter iter = leaf.beginValueOn(); iter; ++iter) {
        Vec3s newVal = *iter + dt * gravity;
        iter.setValue(newVal);
    }
}
    . . .
};// ApplyGravityOp
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

FLIP::PRESSURE PROJECTION

REMOVE DIVERGENCE

- Update ***v****** according to

$$\mathbf{v}^{n+1} = \mathbf{v}^* - \nabla p$$

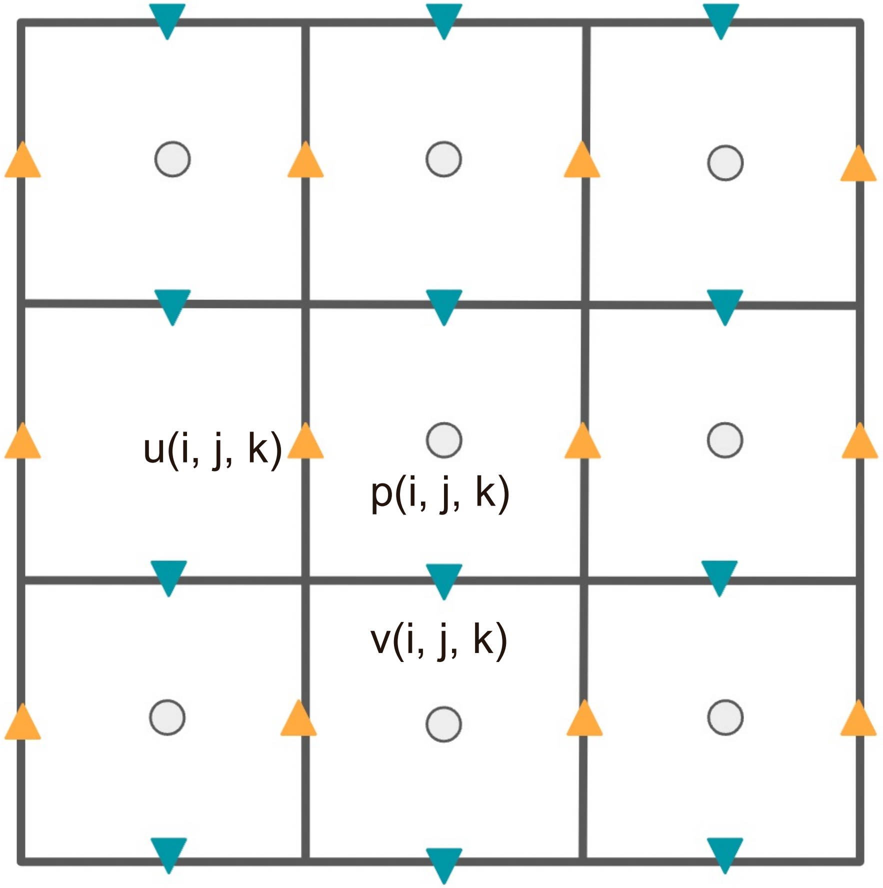
- *p* is solved so that

$$0 = \nabla \cdot \mathbf{v}^{n+1} = \nabla \cdot (\mathbf{v}^* - \nabla p) \iff \nabla^2 p = \nabla \cdot \mathbf{v}^*$$

- Pseudo code

```
// Compute the divergence of the field after adding gravity
// Set up the Poisson equation, pass the divergence of velocity as the right-hand side
// Solve the Poisson equation to get pressure
// Subtract the gradient of pressure from the velocity field
```

STAGGERED/MAC GRID



- Pressure (*i*, *j*, *k*) identifies cell's center
- Each element (*u*, *v*, *w*) at index (*i*, *j*, *k*) is identified with (*i*−1/2, *j*, *k*) face , (*i*, *j*−1/2, *k*) face, and (*i*, *j*, *k*−1/2) face respectively.
- API call

```
vel->setGridClass(GRID_STAGGERED); // Types.h
```

- Other Grid Class

```
enum GridClass {
GRID_UNKNOWN = 0,
GRID_LEVEL_SET,
GRID_FOG_VOLUME,
GRID_STAGGERED
}; // in Types.h
```

COMPUTING DIVERGENCE

- API call (note: want to intersect with interior pressure)

```
mDivBefore = tools::divergence(*mVCurr);
mDivBefore->tree().topologyIntersection(mInteriorPressure->tree());
mDivBefore->setName("div_before");
```

- Library function

```
class Divergence
{
    typename OutGridType::Ptr process(bool threaded = true) {
        if (mInputGrid.getGridClass() == GRID_STAGGERED) {
            Functor<math::FD_1ST> functor(mInputGrid, mMask, threaded, mInterrupt);
            processTypedMap(mInputGrid.transform(), functor);
            return functor.mOutputGrid;
         } else {
            Functor<math::CD_2ND> functor(mInputGrid, mMask, threaded, mInterrupt);
            processTypedMap(mInputGrid.transform(), functor);
            return functor.mOutputGrid;
         }
    }
};
```

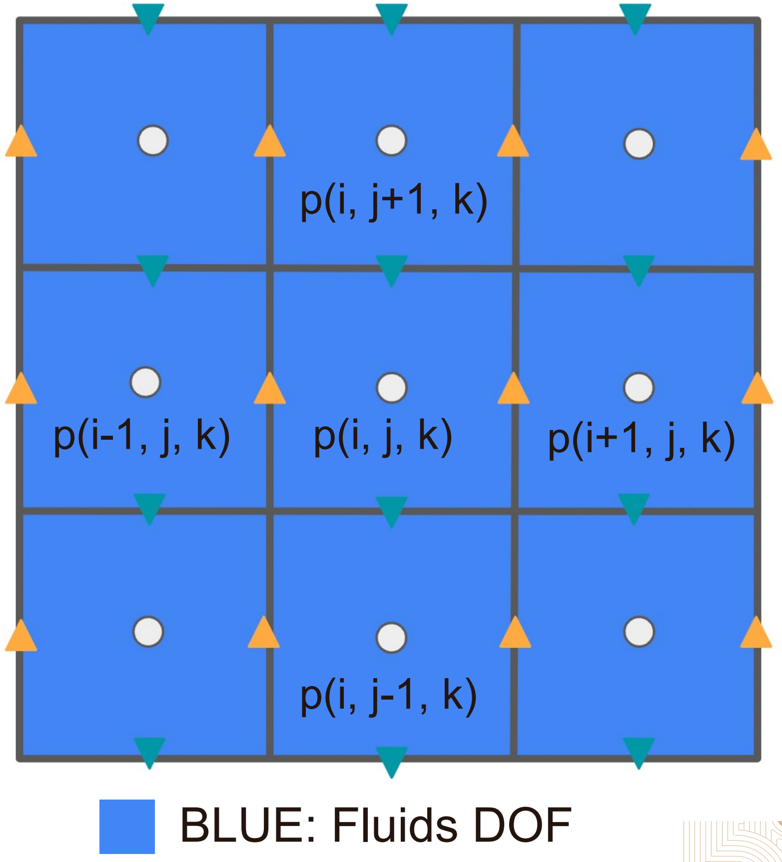
POISSON SOLVE


```cpp
#include <openvdb/tools/PoissonSolver.h> // for poisson solve

{
    using TreeType = FloatTree;
    using ValueType = TreeType::ValueType;
    using MaskGridType = BoolGrid;
    using PCT = openvdb::math::pcg::JacobiPreconditioner<openvdb::tools::poisson::LaplacianMatrix>;
    const double epsilon = math::Delta<ValueType>::value();

    math::pcg::State state = math::pcg::terminationDefaults<ValueType>();
    state.iterations = 100000;
    state.relativeError = state.absoluteError = epsilon;
    FlipSolver::BoundaryOp bop(mVoxelSize, mCollider, mVCurr);
    util::NullInterrupter interrupter;
    FloatTree::Ptr fluidPressure =  tools::poisson::solveWithBoundaryConditionsAndPreconditioner<PCT>
        (mDivBefore->tree(), mInteriorPressure->tree(), bop, state, interrupter, /*staggered=*/true);
}
```

BOUNDARY OPERATOR DEEP DIVE

- Interior discrete equation

$$-6p_{i,j,k} + p_{i-1,j,k} + p_{i+1,j,k} + p_{i,j-1,k} + p_{i,j+1,k} + p_{i,j,k-1} + p_{i,j,k+1} = \text{rhs}_{i,j,k}$$



DIRICHLET (FREE SURFACE) BOUNDARY CONDITION

- What the solver computes out of the box

$$-5p_{i,j,k} + 0 + p_{i+1,j,k} + p_{i,j-1,k} + p_{i,j+1,k} + p_{i,j,k-1} + p_{i,j,k+1} = \text{rhs}_{i,j,k}$$

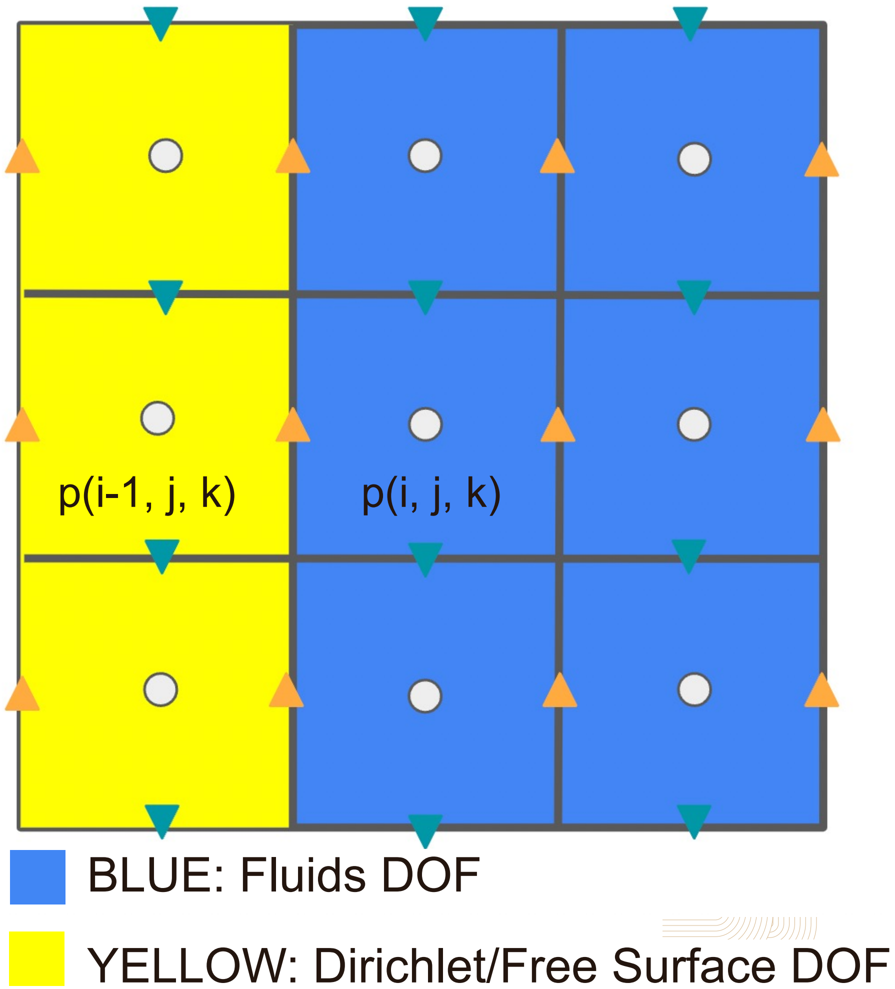
- Given

$$p_{i-1,j,k} = d_{i-1,j,k}$$

- "New" equation

$$-6p_{i,j,k} + 0 + p_{i+1,j,k} + p_{i,j-1,k} + p_{i,j+1,k} + p_{i,j,k-1} + p_{i,j,k+1} = \text{rhs}_{i,j,k} - d_{i-1,j,k}$$

- Subtract 1 from diagonal
- Subtract Dirichlet condition from source



NEUMANN (SOLID) BOUNDARY CONDITION

- What the solver computes out of the box

$$-5p_{i,j,k} + 0 + p_{i+1,j,k} + p_{i,j-1,k} + p_{i,j+1,k} + p_{i,j,k-1} + p_{i,j,k+1} = \text{rhs}_{i,j,k}$$

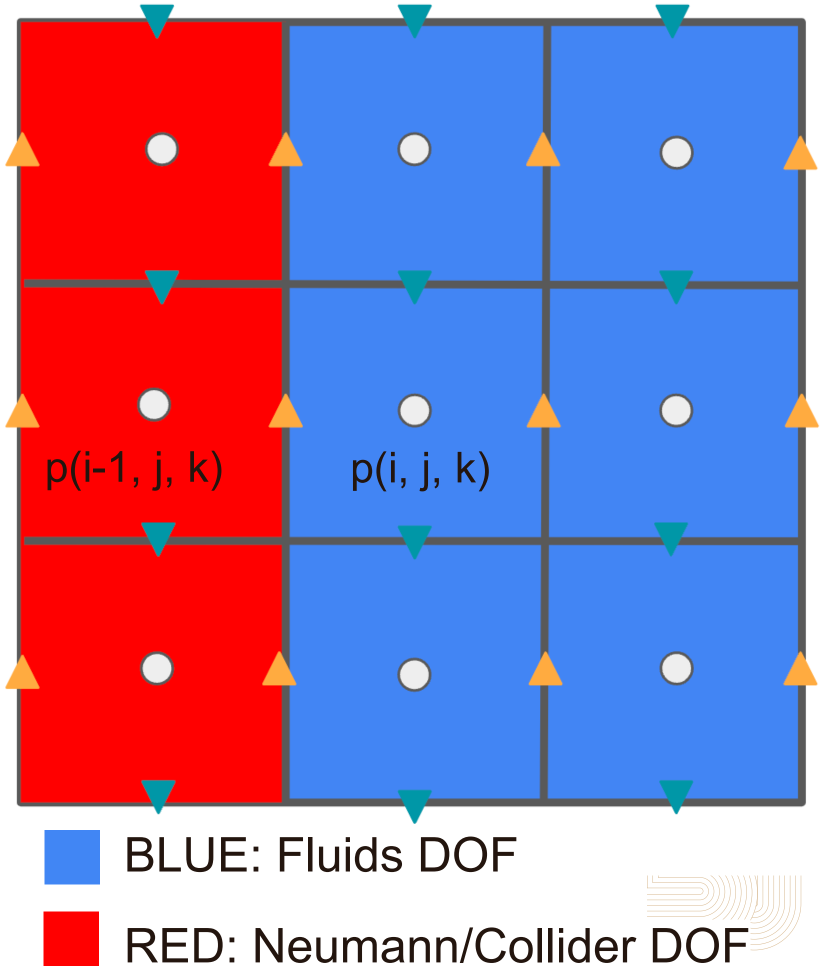
- Given

$$p_{i,j,k} - p_{i-1,j,k} = \Delta x \cdot n_{i-1,j,k}$$

- “New” equation

$$-5p_{i,j,k} + 0 + p_{i+1,j,k} + p_{i,j-1,k} + p_{i,j+1,k} + p_{i,j,k-1} + p_{i,j,k+1} = \text{rhs}_{i,j,k} + \Delta x \cdot n_{i-1,j,k}$$

- Subtract/add voxel size × Neumann condition from source



BOUNDARY OPERATOR FOR POISSON SOLVE


```
void operator()(const openvdb::Coord& ijk,
                const openvdb::Coord& neighbor,
                double& source,
                double& diagonal) const {
    float const dirichletBC = 0.f;
    auto vNgbr = Vec3s(0.f, 0.f, 0.f); // static collider
    bool isInsideCollider = collider->tree().isValueOn(neighbor);

    if (isInsideCollider) {
        double delta = 0.0;
         // Neumann pressure from bbox
         if (neighbor.x() + 1 == ijk.x() /* left x-face */) {
             delta += vNgbr[0]; }
         if (neighbor.x() - 1 == ijk.x() /* right x-face */) {
             delta -= vNgbr[0]; }
         . . .
         source += delta / voxelSize;
    } else /* Dirichlet */ {
         diagonal -= 1.0;
         source -= dirichletBC;
    }
}
```

SUBTRACTING GRADIENT

- Serial pseudocode (see repository for parallel implementation)

```
auto vCurrAcc = mVCurr->getAccessor();
auto vNextAcc = mVNext->getAccessor();
auto interiorAcc = mInteriorPressure->getAccessor();
for (auto iter = mVCurr->beginValueOn(); iter; ++iter) {
    math::Coord ijk = iter.getCoord();
    math::Coord im1jk = ijk.offsetBy(-1, 0, 0);
    math::Coord ijm1k = ijk.offsetBy(0, -1, 0);
    math::Coord ijkm1 = ijk.offsetBy(0, 0, -1);
    // Assumes that mVCurr was set up using mVCurr value
    // Only updates velocity if it is a face of fluid cell
    if (interiorAcc.isValueOn(ijk) || interiorAcc.isValueOn(im1jk) ||
        interiorAcc.isValueOn(ijm1k) || interiorAcc.isValueOn(ijkm1)) {
        Vec3s gradijk;
        gradijk[0] = pressureAcc.getValue(ijk) - pressureAcc.getValue(ijk.offsetBy(-1, 0, 0));
        gradijk[1] = pressureAcc.getValue(ijk) - pressureAcc.getValue(ijk.offsetBy(0, -1, 0));
        gradijk[2] = pressureAcc.getValue(ijk) - pressureAcc.getValue(ijk.offsetBy(0, 0, -1));
        auto val = vCurrAcc.getValue(ijk) - gradijk * mVoxelSize;
        vNextAcc.setValue(ijk, val);
    }
}
```

THE PREMIER CONFERENCE & EXHIBITION ON
COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

FLIP::GRID TO PARTICLES

GRID TO PARTICLES, A.K.A. INTERPOLATION


```
void
FlipSolver::gridToParticles() {
    // Interpolate PIC velocity
    points::boxSample(*mPoints, *mVNext, "v_pic");

    // Interpolate FLIP velocity
    points::boxSample(*mPoints, *mVDiff, "v_flip");
}
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

FLIP::UPDATE PARTICLES

PARTICLE VELOCITY UPDATE

- Update particle velocity (solving advection equation in a Lagrangian manner)

$$\mathbf{v}_p^{\mathrm{PIC}} = \sum_i \mathbf{v}_i^{n+1} N_i(\mathbf{x}_p^n)$$

$$\mathbf{v}_p^{\mathrm{FLIP}} = \sum_i (\mathbf{v}_i^{n+1} - \mathbf{v}_i^n) N_i(\mathbf{x}_p^n)$$

$$\mathbf{v}_p^{n+1} = \alpha(\mathbf{v}_p^n + \mathbf{v}_p^{\mathrm{FLIP}}) + (1-\alpha)\mathbf{v}_p^{\mathrm{PIC}}$$

- Implement using Leaf Manager

```
tree::LeafManager<points::PointDataTree> leafManager(mPoints->tree());
FlipSolver::FlipUpdateOp op(velIdx, vPicIdx, vFlipIdx, 0.9f /* alpha in PIC/FlIP update */);
tbb::parallel_for(leafManager.leafRange(), op);
```

PARTICLE VELOCITY UPDATE IMPLEMENTATION


```
void operator()(const tree::LeafManager<points::PointDataTree>::LeafRange& range) const {
    for (auto leafIter = range.begin(); leafIter; ++leafIter) {
        points::AttributeArray& velArray = leafIter->attributeArray(velAtrIdx);
        points::AttributeArray const& vPicArray = leafIter->constAttributeArray(vPicAtrIdx);
        points::AttributeArray const& vFlipArray = leafIter->constAttributeArray(vFlipAtrIdx);
        points::AttributeWriteHandle<Vec3s> velHandle(velArray);
        points::AttributeHandle<Vec3s> vPicHandle(vPicArray);
        points::AttributeHandle<Vec3s> vFlipHandle(vFlipArray);
        // Iterate over active indices in the leaf.
        for (auto indexIter = leafIter->beginIndexOn(); indexIter; ++indexIter) {
            auto curVel = velHandle.get(*indexIter);
            auto vPic = vPicHandle.get(*indexIter);
            auto vFlip = vFlipHandle.get(*indexIter);
            auto newVel = alpha * (curVel + vFlip) + (1.f - alpha) * vPic;
            velHandle.set(*indexIter, newVel);
        }
    }
}
```

UPDATE PARTICLE POSITION


```
void
    FlipSolver::advectParticles(float const dt) {
    Index const integrationOrder = 4; // RK4
    int const steps = 1;
    points::advectPoints(*mPoints, *mVNext, integrationOrder, dt, steps);
}
```

THE PREMIER CONFERENCE & EXHIBITION ON
COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

CODA

LESSON LEARNED

- Setting up Poisson solve in OpenVDB, working with MAC grid.
- Being careful with the topology of the domain is crucial.
- Sparse grid implementation requires extrapolation.
- Often good workflow: define the topology of your grid first, then fill in the values.
- Not covered: how to do extrapolation and optimization methods (e.g. using memory pool instead of allocating new grids at each step).

THANK YOU

- Thanks to Greg Klár and Dan Bailey for their help and for answering many questions
- Thanks to Ken Museth, Jeff Lait, Nick Avramoussis, Rich Jones, Jeff Budsberg, Greg Hurst

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

THANK YOU