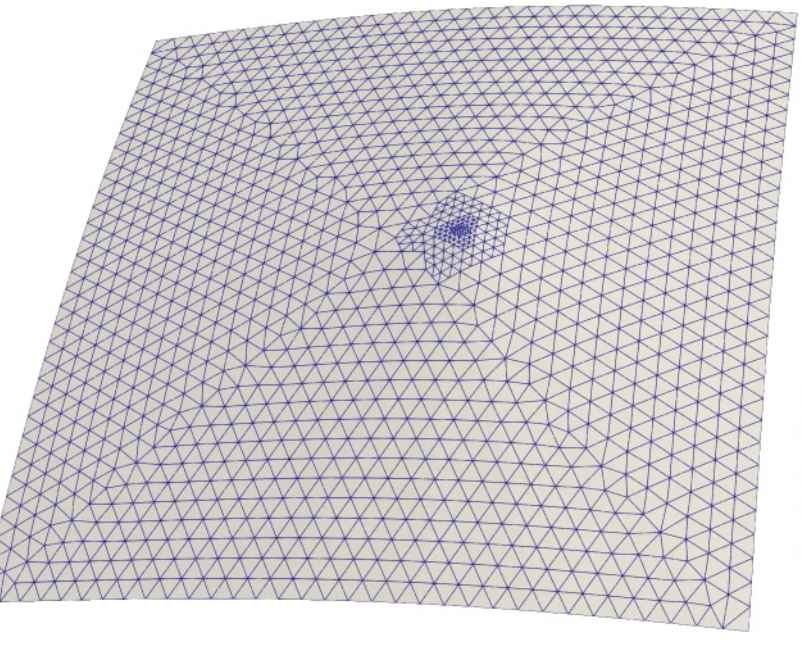
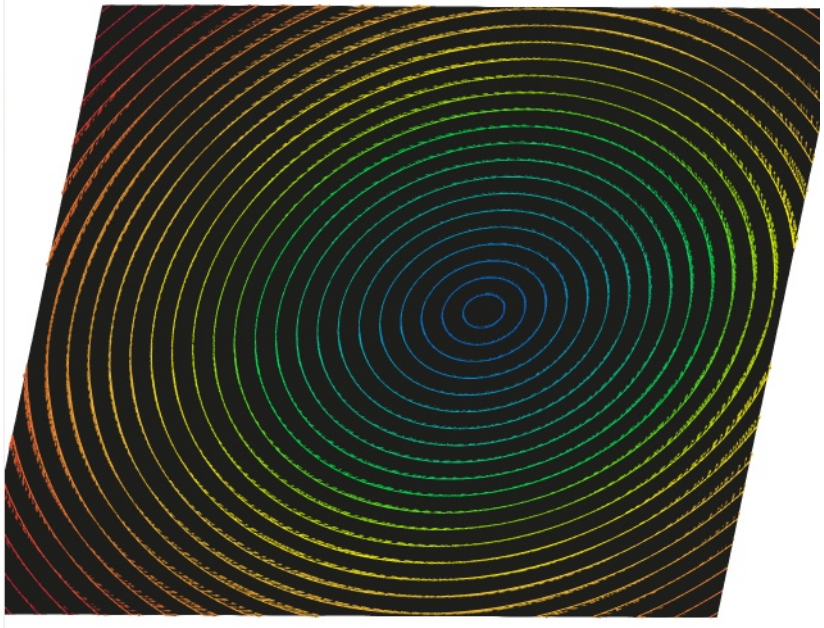
Advection Problem

Transport equation



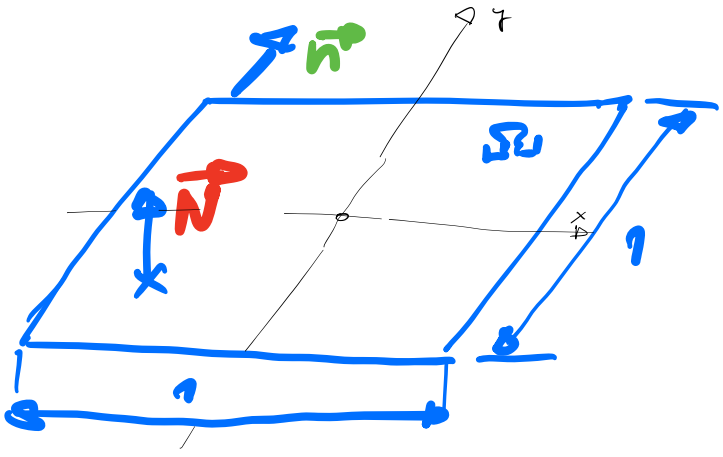


* Problem setup
* Upwind Discontinuous Galerkin
* Non conforming mesh refinement
* Error indicator and h-adaptivity
* Bit reference levels

Problem Setup

Advection problem

$$\frac{\partial \varphi}{\partial t} + \nabla \cdot (\varphi \cdot \vec{v}) = 0 \quad \text{in} \quad \Omega$$



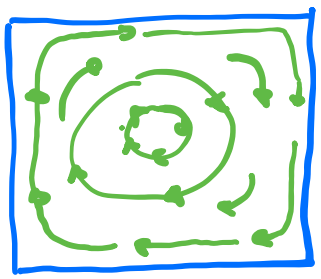
Velocity potential

$$\vec{v} = (\vec{\nabla} \times \vec{\phi}), \quad \phi = \theta \vec{N}$$

$$\vec{v} = \{ 1 - \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial x}, 0 \}$$

$$v_i = \varepsilon_{ij} \frac{\partial \phi}{\partial x_j} = \varepsilon_{ij} \phi_{,j}$$

$$\varepsilon_{ij} = \begin{cases} 0 & i = j \\ -1 & i = 0 \\ 1 & i = 1 \end{cases}$$ Levi-civita symbol



$$v_i n_i = 0$$

zero normal

$$\nabla \cdot \vec{\nabla} \times \phi = 0$$

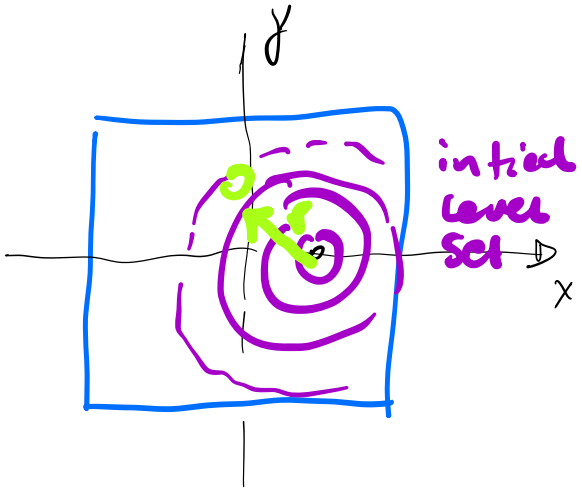
Divergence free

Initial level set

$$x_c = \{ 1/10, 0 \} \qquad r = \frac{1}{5}$$

$$\Delta x = x - x_c$$

$$\varphi = \sqrt{\Delta x \cdot \Delta x} = \sqrt{\Delta x_i \Delta x_i}$$



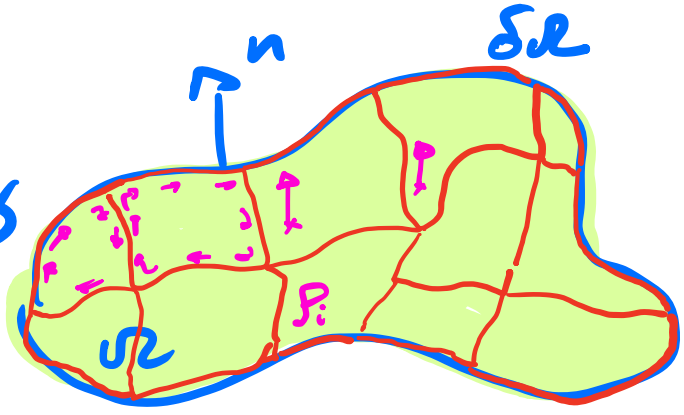
Upwind Discontinuous Galerkin

$$\frac{\partial \varphi}{\partial t} + \nabla \cdot (\varphi v_i) = 0 \quad / \delta\varphi / \int_\Omega$$

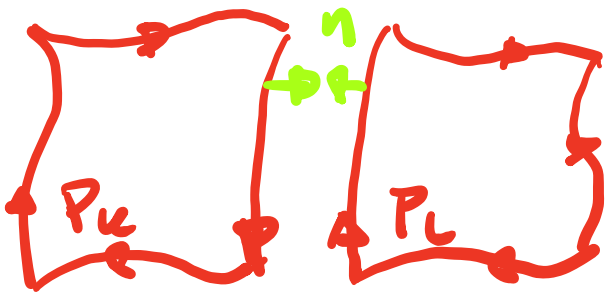
$$\int_\Omega \delta\varphi \frac{\partial \varphi}{\partial t} d\Omega + \int_\Omega \delta\varphi \nabla \cdot (\varphi v_i) d\Omega = 0$$

$$\int_\Omega \delta\varphi \frac{\partial \varphi}{\partial t} d\Omega - \int_\Omega \delta\varphi_{,i} v_i \varphi \, d\Omega + \int_{\partial\Omega} \delta\varphi \, n_i v_i \varphi \, d\delta\Omega = 0$$

$$\delta\Omega = \bigcup_k P_k \qquad P_k \cap P_l = \emptyset \text{ if } k \neq l$$

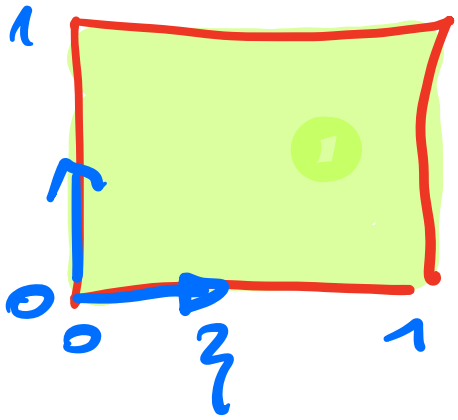


$$\sum_k \int_{P_k} \delta\varphi \frac{\partial \varphi}{\partial t} dP_k - \sum_k \int_{P_k} \delta\varphi_{,i} v_i \varphi \, dP_k + \sum_k \int_{\partial P_k} \delta\varphi \cdot n_i v_i \varphi \, d\delta P_k = 0$$



Discretisation in space

$\varphi \approx \varphi^h \quad \varphi^h \in P^h$

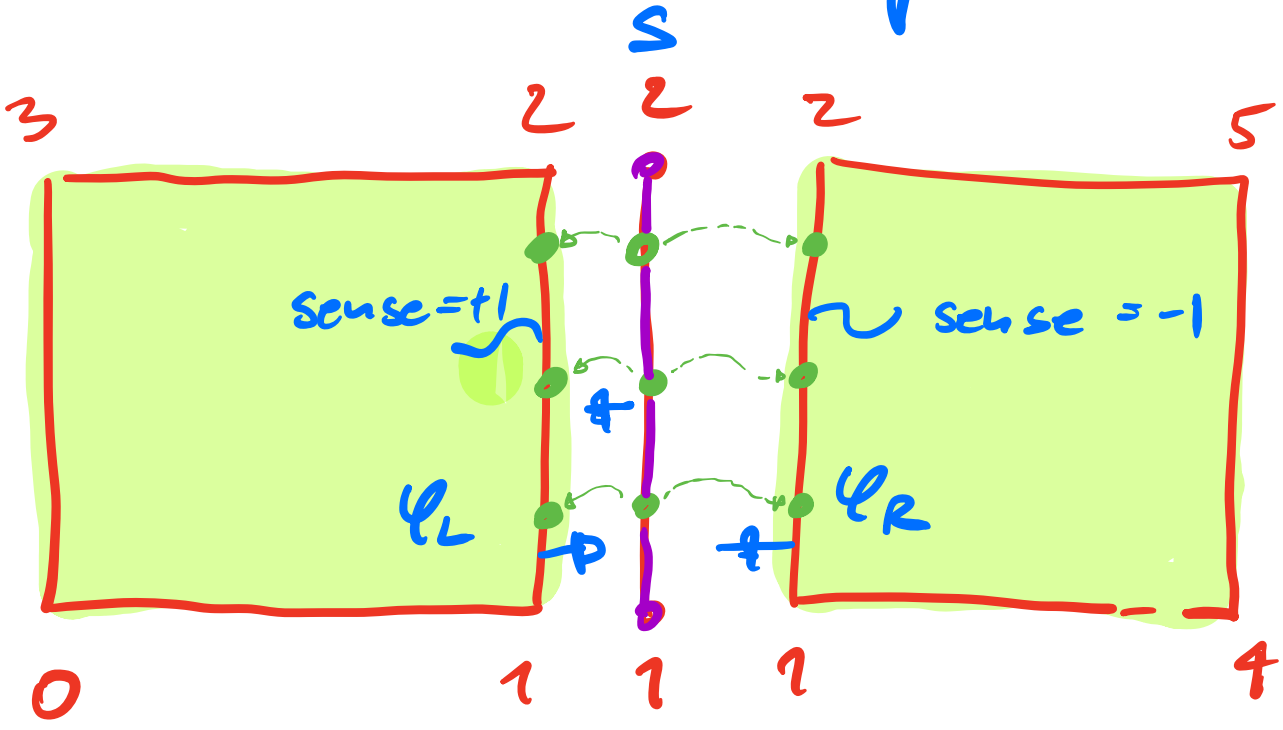


$$\varphi^h(\xi,\eta) = \sum_k \sum_l L_k(\xi) L_l(\eta)\, c_{kl}$$

Legendre Polynomial

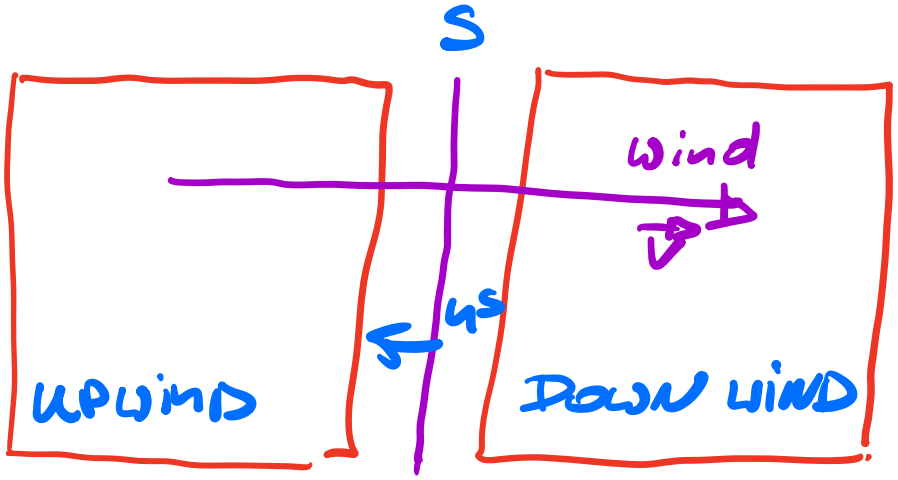
$$\int_0^1 L_k(\xi) L_l(\xi)\, d\xi = 0$$

orthogonal



$\varphi_L^h \neq \varphi_R^h$

Upwinding



$$\varphi_L^k = \varphi_{up}^k$$

$$\sum_{P_k} \int_{P_k} \left(\delta\varphi^k \frac{\partial \varphi^k}{\partial t} - \delta\varphi_{,i}^k v_i \varphi^k \right) d P_k$$

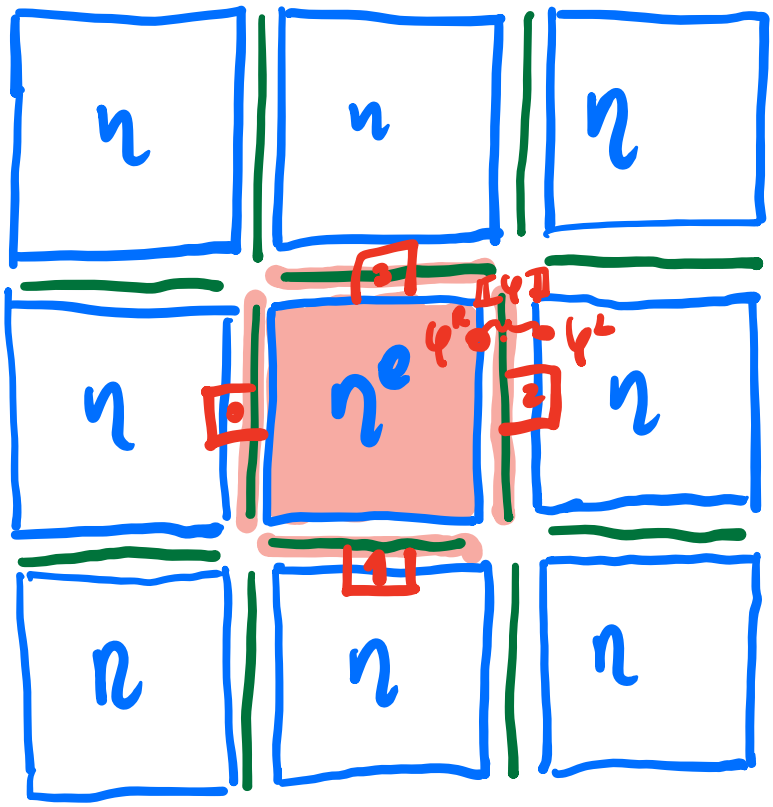
$$+ \sum_{S} \int_{S} [\![\delta\varphi]\!] \langle v n \rangle \varphi_{up}^k \, dS = 0$$

$$[\![\delta\varphi^k]\!] = \delta\varphi_R^k - \delta\varphi_L^k$$

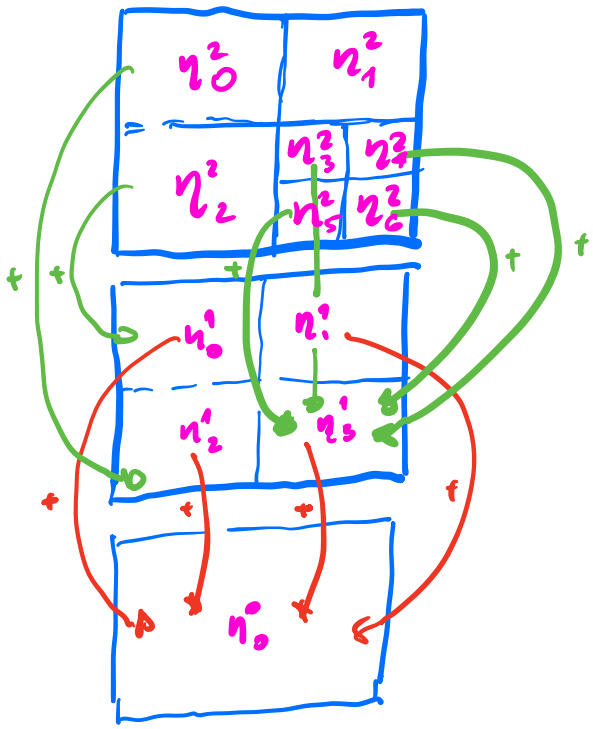
$$\langle v n \rangle = v n^R - v n^L$$

+ discretisation in time

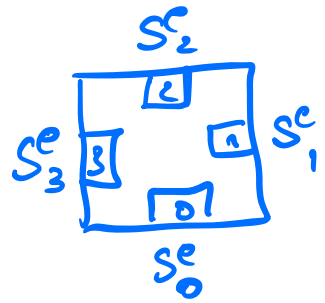
Error indicator

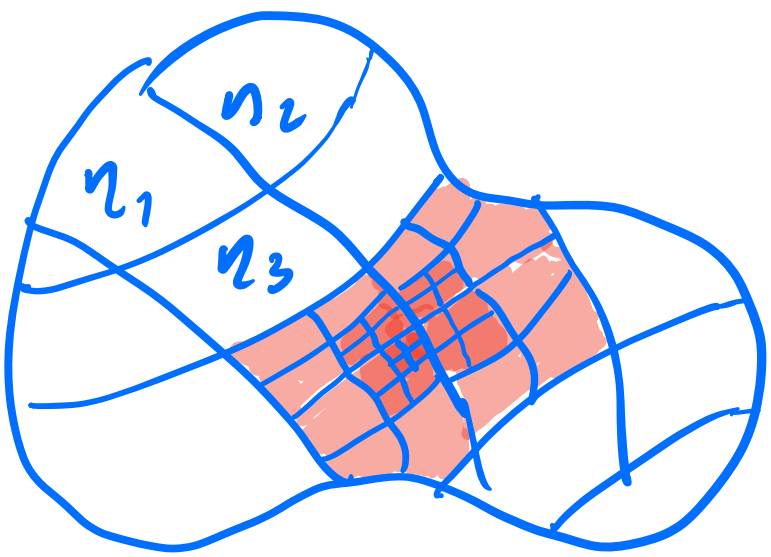


Projection of error from fine to coarse mesh



$$\eta^e = \int_{S^e} [\![\varphi]\!] \, dS^e = \sum_i \int_{S_i^e} (\varphi^L - \varphi^R) \, dS_i^e$$



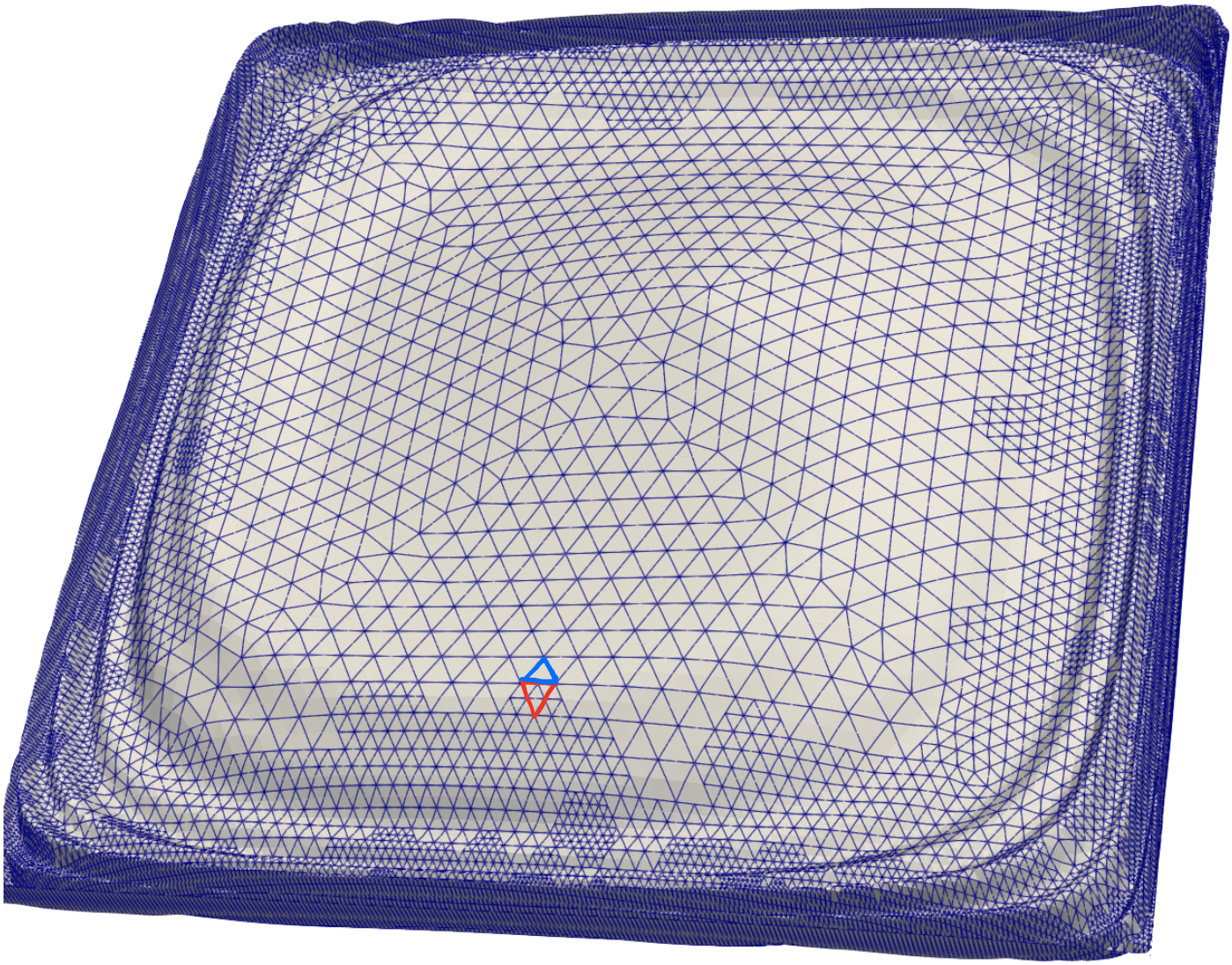


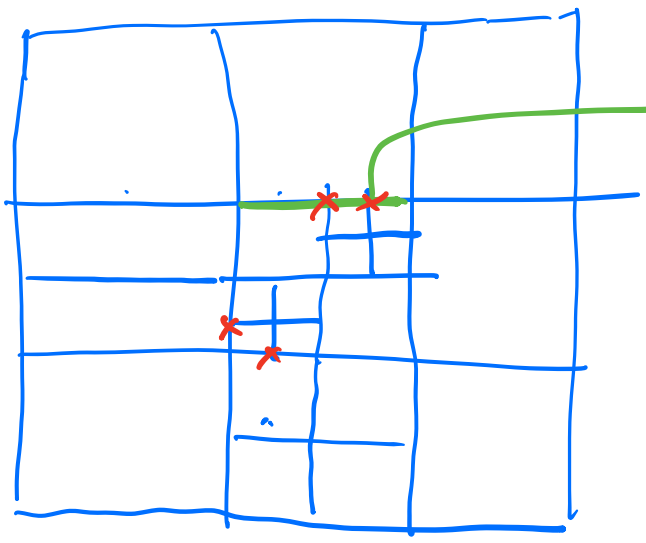
Elements to refine

$$\eta_e > \varepsilon \max_e(\eta^e)$$

ε threshold

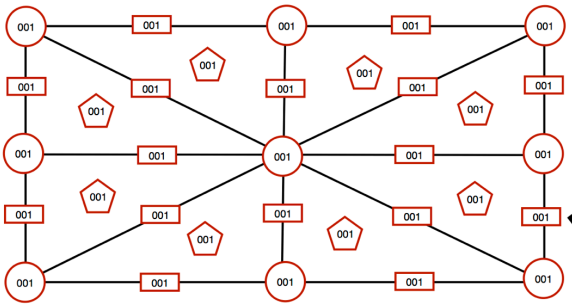
Topological restrictions



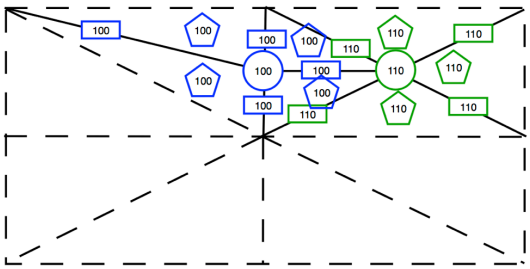


Look to COR-1 tutorial to learn about bit ref levels

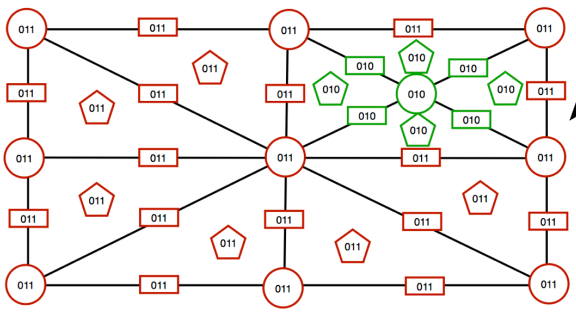
Refined Bit Level 001 and Mask 111



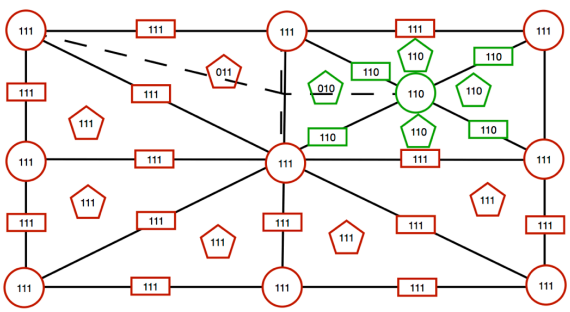
Seclection Bit Level 100 and Mask 110



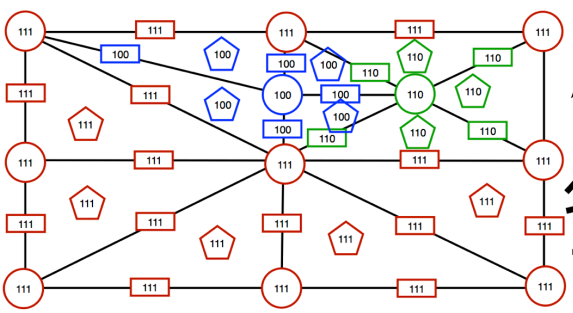
Refined Bit Level 010 and Mask 111



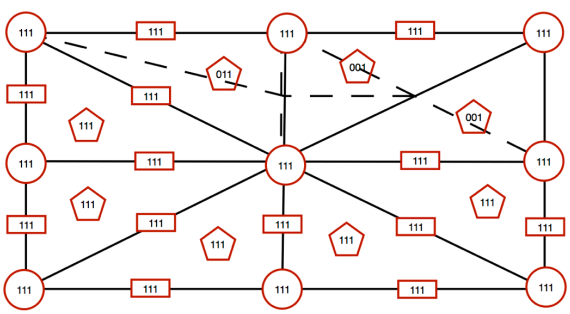
Selection Bit Level 010 and Mask 111

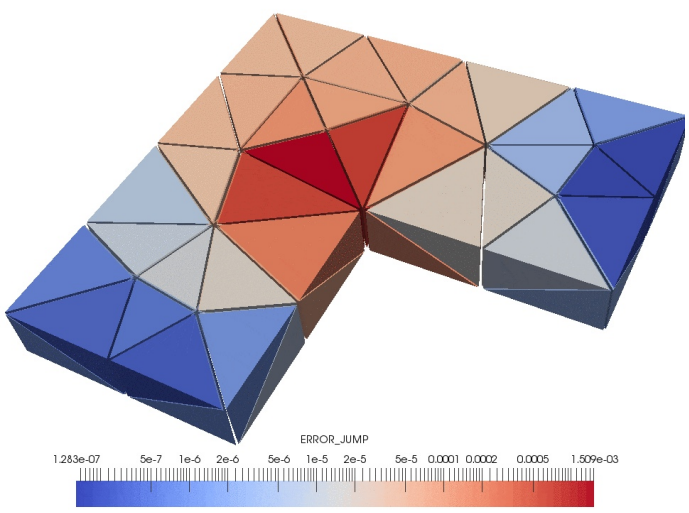


Refined Bit Level 100 and Mask 111

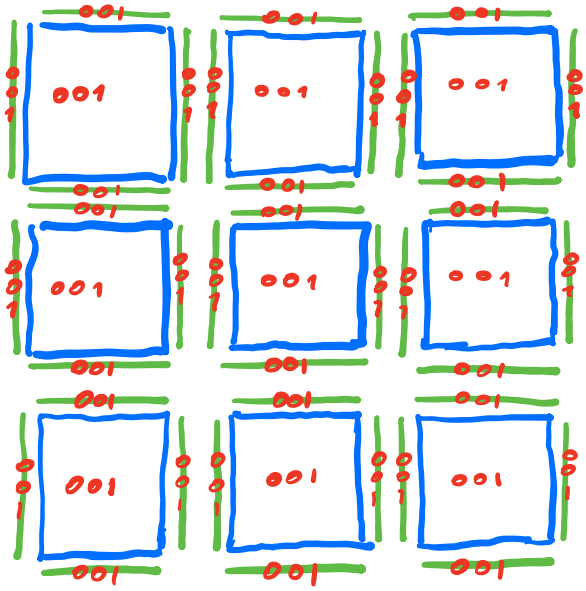


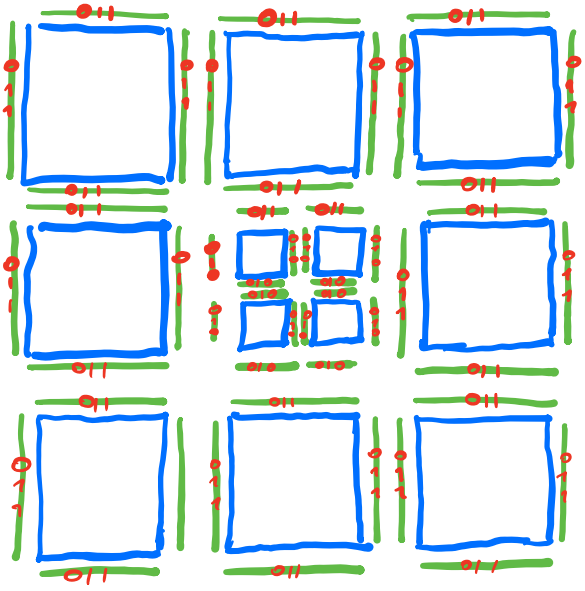
Selection Bit Level 001 and Mask 111

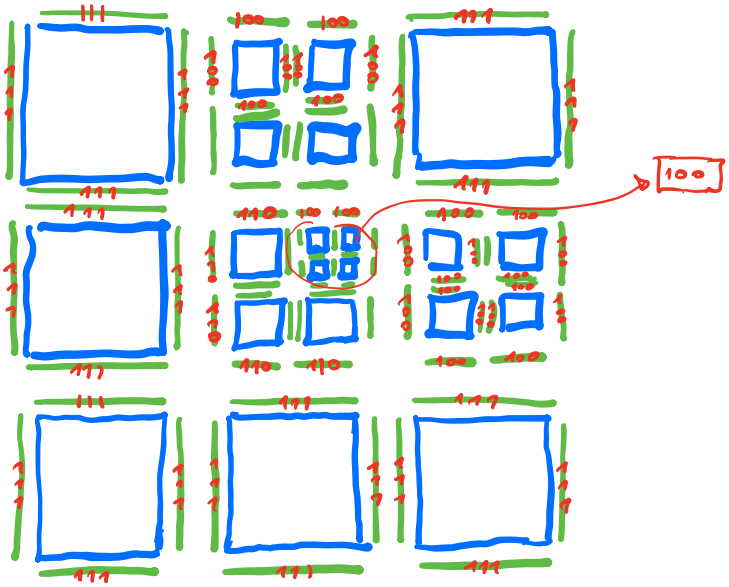




Bit reference levels







bit = 001
010
100
110
011

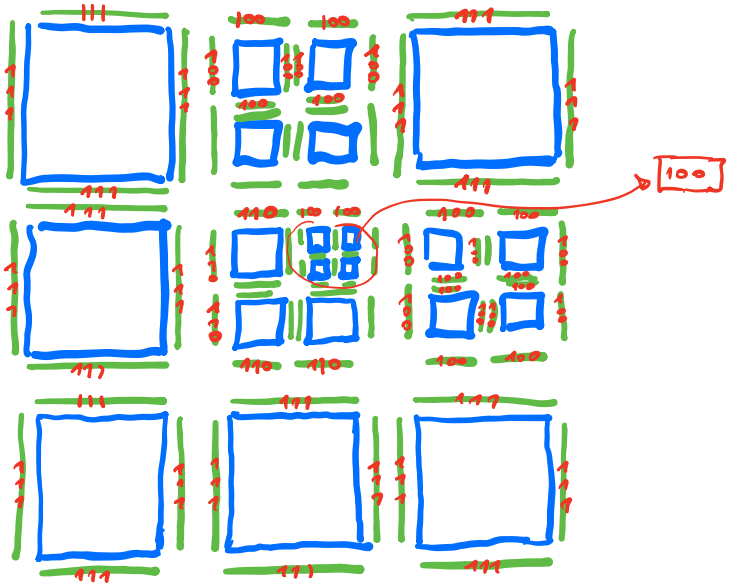
mask = 001
010
100

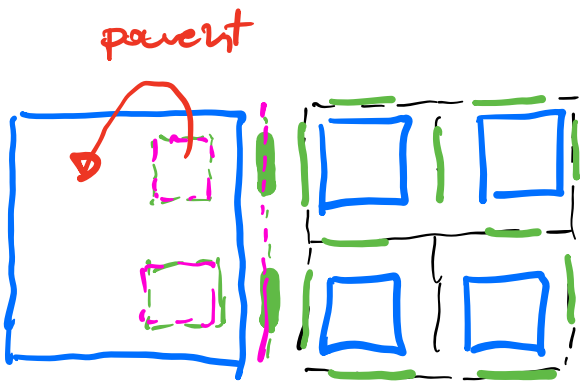
bit & mask

Integration over skeleton

$$\int_S [\![\varphi]\!] \, \mathrm{curv}^k \, \varphi_{up}^k \, dS$$

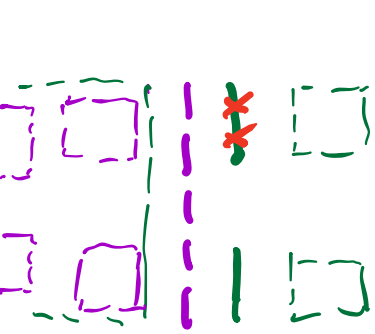
S S





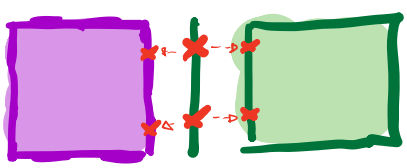
① Skeleton

② Take adj entities from current skeleton faces

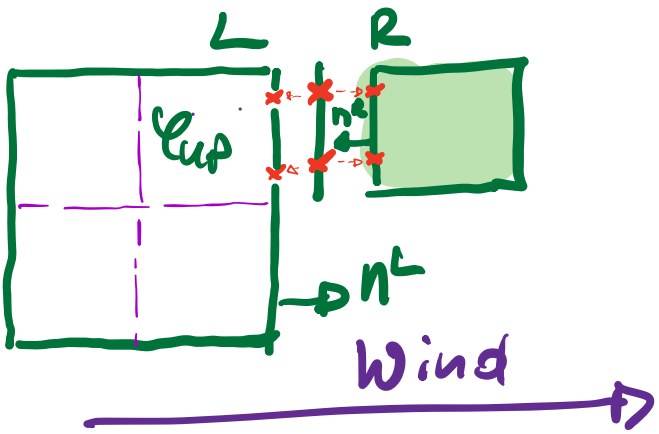


skeleton bit

current bit



(3) Take parent of skeleton bit adjacent entities.



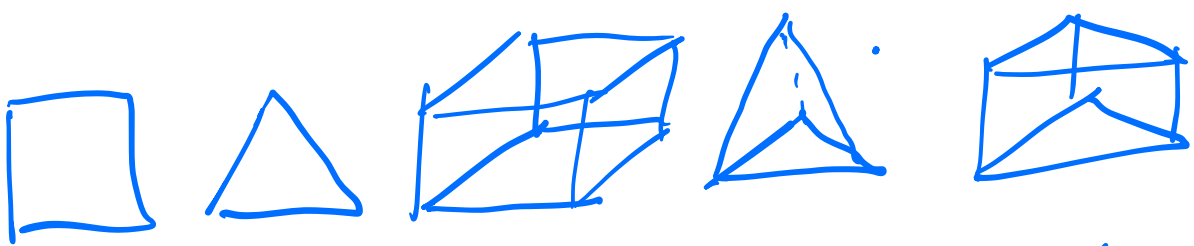
$$\int_S [\![\delta\varphi]\!] \langle v \cdot n \rangle \varphi_{up} \, dS$$

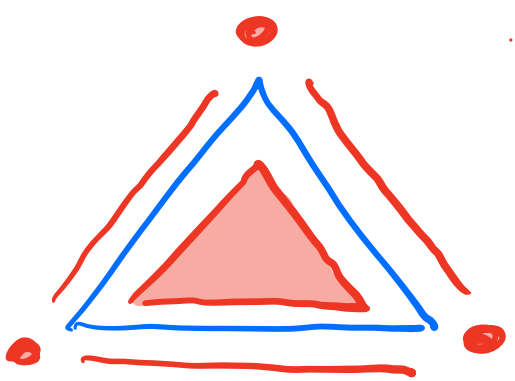
$$[\![\delta\varphi]\!] = \delta\varphi^L - \delta\varphi^R$$

$$\langle v \cdot n \rangle = v \cdot n^L - v \, n^R$$

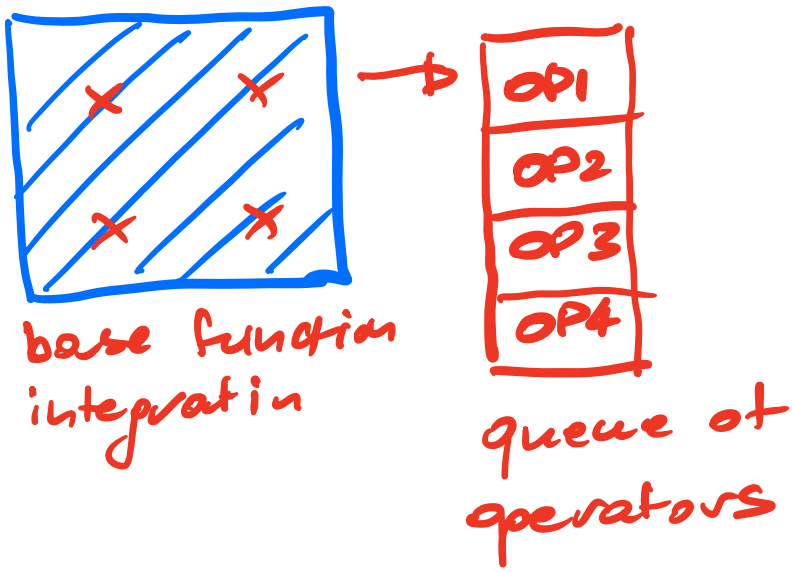
User Data Operators & Finite element instances.

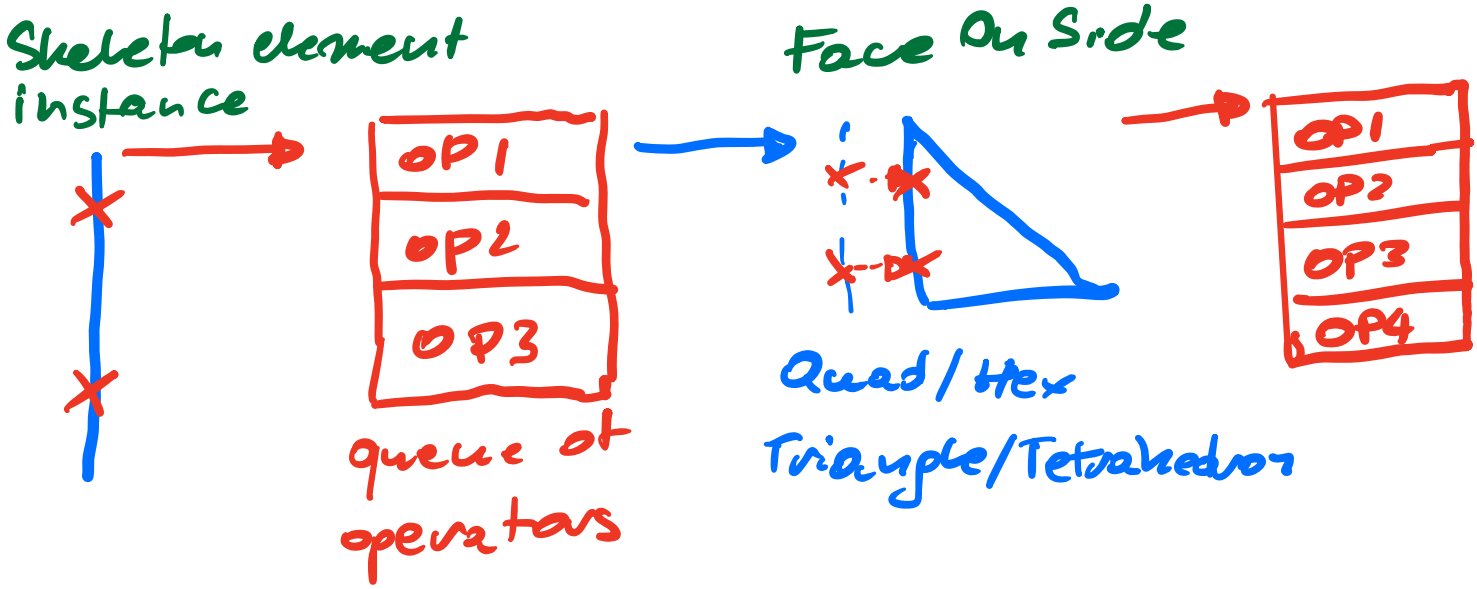
Finite element can be any shape, any type, skeleton or domain. Finite element instance provides integration rule and base functions.

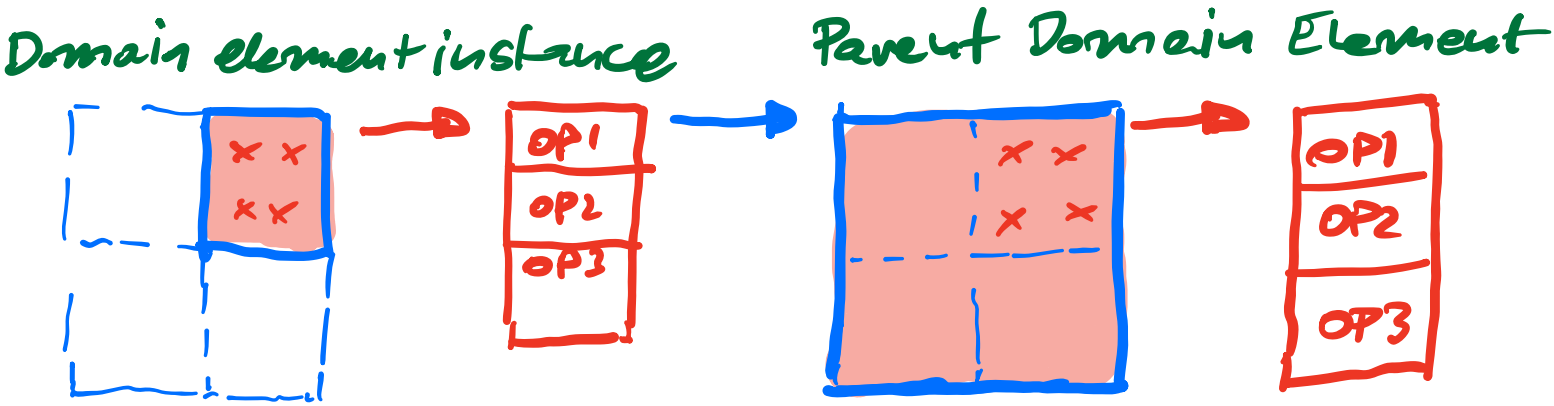




Element is broken on entities, on each entity is executed operator. Operator evaluate base functions and DOFs associated with particular field & base


OP1
OP2
OP3
OP4
base function integration
queue of operators


Skeleton element instance
OP1
OP2
OP3
queue of operators
Face On Side
Quad / Hex
Triangle/Tetrahedron
OP1
OP2
OP3
OP4


Domain element instance
OP1
OP2
OP3
Parent Domain Element
OP1
OP2
OP3

```cpp
/**
 * @file level_set.cpp
 * @example level_set.cpp
 * @brief Implementation DG upwind method for advection/level set problem
 * @date 2022-12-15
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <MoFEM.hpp>
using namespace MoFEM;

static char help[] = "...\n\n";

//! [Define dimension]
constexpr int SPACE_DIM = EXECUTABLE_DIMENSION;
constexpr AssemblyType A = AssemblyType::PETSC; //< selected assembly type
constexpr IntegrationType I =
    IntegrationType::GAUSS; //< selected integration type

using EntData = EntitiesFieldData::EntData;
using DomainEle = PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::DomainEle;
using DomianParentEle =
    PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::DomianParentEle;
using BoundaryEle =
    PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::BoundaryEle;
using FaceSideEle =
    PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::FaceSideEle;

using DomainEleOp = DomainEle::UserDataOperator;
using BoundaryEleOp = BoundaryEle::UserDataOperator;
using FaceSideEleOp = FaceSideEle::UserDataOperator;

using PostProcEle = PostProcBrokenMeshInMoab<DomainEle>;

constexpr FieldSpace potential_velocity_space = SPACE_DIM == 2 ? H1 : HCURL;
constexpr size_t potential_velocity_field_dim = SPACE_DIM == 2 ? 1 : 3;

constexpr bool debug = false;
constexpr int nb_levels = 3;

constexpr int start_bit = nb_levels + 1;

constexpr int current_bit = 2 * start_bit + 1;
constexpr int skeleton_bit = 2 * start_bit + 2;
constexpr int aggregate_bit = 2 * start_bit + 3;
constexpr int projection_bit = 2 * start_bit + 4;
constexpr int aggregate_projection_bit = 2 * start_bit + 5;

struct LevelSet {

  LevelSet(MoFEM::Interface &m_field) : mField(m_field) {}

  MoFEMErrorCode runProblem();

private:
  using VecSideArray = std::array<VectorDouble, 2>;
  using MatSideArray = std::array<MatrixDouble, 2>;

  /**
   * @brief data structure carrying information on skeleton on both sides.
   *
   */
  struct SideData {
    // data for skeleton computation
    std::array<EntityHandle, 2> feSideHandle;
    std::array<VectorInt, 2>
        indicesRowSideMap; ///< indices on rows for left hand-side
    std::array<VectorInt, 2>
        indicesColSideMap; ///< indices on columns for left hand-side
    std::array<MatrixDouble, 2> rowBaseSideMap; // base functions on rows
    std::array<MatrixDouble, 2> colBaseSideMap; // base function  on columns
    std::array<int, 2> senseMap; // orientation of local element edge/face in
                                 // respect to global orientation of edge/face

    VecSideArray lVec;   //< Values of level set field
    MatSideArray velMat; //< Values of velocity field

    int currentFESide; ///< current side counter
  };

  /**
   * @brief advection velocity field
   *
   * \note in current implementation is assumed that advection field has zero
```

Number of refinement levels.

```
   * normal component.
   *
   * \note function define a vector velocity potential field, curl of potential
   * field gives velocity, thus velocity is divergence free.
   *
   * @tparam SPACE_DIM
   * @param x
   * @param y
   * @param z
   * @return auto
   */
  template <int SPACE_DIM>
  static double get_velocity_potential(double x, double y, double z);

  /**
   * @brief inital level set, i.e. advected filed
   *
   * @param x
   * @param y
   * @param z
   * @return double
   */
  static double get_level_set(const double x, const double y, const double z);

  /**
   * @brief read mesh
   *
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode readMesh();

  /**
   * @brief create fields, and set approximation order
   *
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode setupProblem();

  /**
   * @brief push operators to integrate operators on domain
   *
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode pushOpDomain();

  /**
   * @brief evaluate error
   *
   * @return MoFEMErrorCode
   */
  std::tuple<double, Tag> evaluateError();

  /**
   * @brief Get operator calculating velocity on coarse mesh
   *
   * @param vel_ptr
   * @return DomainEleOp*
   */
  ForcesAndSourcesCore::UserDataOperator *
  getZeroLevelVelOp(boost::shared_ptr<MatrixDouble> vel_ptr);

  /**
   * @brief create side element to assemble data from sides
   *
   * @param side_data_ptr
   * @return boost::shared_ptr<FaceSideEle>
   */
  boost::shared_ptr<FaceSideEle>
  getSideFE(boost::shared_ptr<SideData> side_data_ptr);

  /**
   * @brief push operator to integrate on skeleton
   *
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode pushOpSkeleton();

  /**
   * @brief test integration side elements
   *
   * Check consistency between volume and skeleton integral.
   *
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode testSideFE();

```

```
  /**
   * @brief test consistency between tangent matrix and the right hand side
   * vectors
   *
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode testOp();

  /**
   * @brief initialise field set
   *
   * @param level_fun
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode initialiseFieldLevelSet(
      boost::function<double(double, double, double)> level_fun =
          get_level_set);

  /**
   * @brief initialise potential velocity field
   *
   * @param vel_fun
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode initialiseFieldVelocity(
      boost::function<double(double, double, double)> vel_fun =
          get_velocity_potential<SPACE_DIM>);

  /**
   * @brief dg level set projection
   *
   * @param prj_bit
   * @param mesh_bit
   * @return MoFEMErrorCode
   */
  MoFEMErrorCode dgProjection(const int prj_bit = projection_bit);

  /**
   * @brief solve advection problem
   *
   * @return * MoFEMErrorCode
   */
  MoFEMErrorCode solveAdvection();

  /**
   * @brief Wrapper executing stages while mesh refinement
   */
  struct WrapperClass {
    WrapperClass() = default;
    virtual MoFEMErrorCode setBits(LevelSet &level_set, int l) = 0;
    virtual MoFEMErrorCode runCalcs(LevelSet &level_set, int l) = 0;
    virtual MoFEMErrorCode setAggregateBit(LevelSet &level_set, int l) = 0;
    virtual double getThreshold(const double max) = 0;
  };

  /**
   * @brief Used to execute inital mesh approximation while mesh refinment
   *
   */
  struct WrapperClassInitalSolution : public WrapperClass {

    WrapperClassInitalSolution(boost::shared_ptr<double> max_ptr)
        : WrapperClass(), maxPtr(max_ptr) {}

    MoFEMErrorCode setBits(LevelSet &level_set, int l) {
      MoFEMFunctionBegin;
      auto simple = level_set.mField.getInterface<Simple>();
      simple->getBitRefLevel() =
          BitRefLevel().set(skeleton_bit) | BitRefLevel().set(aggregate_bit);
      simple->getBitRefLevelMask() = BitRefLevel().set();
      simple->reSetUp(true);
      MoFEMFunctionReturn(0);
    };

    MoFEMErrorCode runCalcs(LevelSet &level_set, int l) {
      MoFEMFunctionBegin;
      CHKERR level_set.initialiseFieldLevelSet();
      MoFEMFunctionReturn(0);
    }

    MoFEMErrorCode setAggregateBit(LevelSet &level_set, int l) {
      auto bit_mng = level_set.mField.getInterface<BitRefManager>();
      auto set_bit = [](auto l) { return BitRefLevel().set(l); };
      MoFEMFunctionBegin;
      Range level;
      CHKERR bit_mng->getEntitiesByRefLevel(set_bit(start_bit + l),
```

```
                                          BitRefLevel().set(), level);
      CHKERR level_set.mField.getInterface<CommInterface>()
          ->synchroniseEntities(level);
      CHKERR bit_mng->setNthBitRefLevel(current_bit, false);
      CHKERR bit_mng->setNthBitRefLevel(level, current_bit, true);
      CHKERR bit_mng->setNthBitRefLevel(level, aggregate_bit, true);
      MoFEMFunctionReturn(0);
    }

    double getThreshold(const double max) {
      *maxPtr = std::max(*maxPtr, max);
      return 0.05 * (*maxPtr);
    }

    private:
      boost::shared_ptr<double> maxPtr;
  };

  /**
   * @brief Use peculated errors on all levels while mesh projection
   *
   */
  struct WrapperClassErrorProjection : public WrapperClass {
      WrapperClassErrorProjection(boost::shared_ptr<double> max_ptr)
          : maxPtr(max_ptr) {}

      MoFEMErrorCode setBits(LevelSet &level_set, int l) { return 0; };
      MoFEMErrorCode runCalcs(LevelSet &level_set, int l) { return 0; }
      MoFEMErrorCode setAggregateBit(LevelSet &level_set, int l) {
      auto bit_mng = level_set.mField.getInterface<BitRefManager>();
      auto set_bit = [](auto l) { return BitRefLevel().set(l); };
      MoFEMFunctionBegin;
      Range level;
      CHKERR bit_mng->getEntitiesByRefLevel(set_bit(start_bit + l),
                                          BitRefLevel().set(), level);
      CHKERR level_set.mField.getInterface<CommInterface>()
          ->synchroniseEntities(level);
      CHKERR bit_mng->setNthBitRefLevel(current_bit, false);
      CHKERR bit_mng->setNthBitRefLevel(level, current_bit, true);
      CHKERR bit_mng->setNthBitRefLevel(level, aggregate_bit, true);
      MoFEMFunctionReturn(0);
    }
    double getThreshold(const double max) { return 0.05 * (*maxPtr); }

  private:
    boost::shared_ptr<double> maxPtr;

  };

  MoFEMErrorCode refineMesh(WrapperClass &&wp);

  struct OpRhsDomain;   ///< integrate volume operators on rhs
  struct OpLhsDomain;   ///< integrate volume operator on lhs
  struct OpRhsSkeleton; ///< integrate skeleton operators on rhs
  struct OpLhsSkeleton; ///< integrate skeleton operators on khs

  // Main interfaces
  MoFEM::Interface &mField;

  using AssemblyDomainEleOp =
      FormsIntegrators<DomainEleOp>::Assembly<A>::OpBase;
  using OpMassLL =
      FormsIntegrators<DomainEleOp>::Assembly<A>::BiLinearForm<I>::OpMass<1, 1>;
  using OpSourceL =
      FormsIntegrators<DomainEleOp>::Assembly<A>::LinearForm<I>::OpSource<1, 1>;
  using OpMassVV = FormsIntegrators<DomainEleOp>::Assembly<A>::BiLinearForm<
      I>::OpMass<potential_velocity_field_dim, potential_velocity_field_dim>;
  using OpSourceV = FormsIntegrators<DomainEleOp>::Assembly<A>::LinearForm<
      I>::OpSource<potential_velocity_field_dim, potential_velocity_field_dim>;
  using OpScalarFieldL = FormsIntegrators<DomainEleOp>::Assembly<A>::LinearForm<
      I>::OpBaseTimesScalar<1>;

  using AssemblyBoundaryEleOp =
      FormsIntegrators<BoundaryEleOp>::Assembly<A>::OpBase;

  enum ElementSide { LEFT_SIDE = 0, RIGHT_SIDE = 1 };

private:
  boost::shared_ptr<double> maxPtr;

};

template <>
double LevelSet::get_velocity_potential<2>(double x, double y, double z) {
  return (x * x - 0.25) * (y * y - 0.25);
}
```

```
345
346 double LevelSet::get_level_set(const double x, const double y, const double z) {
347   constexpr double xc = 0.1;
348   constexpr double yc = 0.;
349   constexpr double zc = 0.;
350   constexpr double r = 0.2;
351   return std::sqrt(pow(x - xc, 2) + pow(y - yc, 2) + pow(z - zc, 2)) - r;
352 }
353
354 MoFEMErrorCode LevelSet::runProblem() {
355   MoFEMFunctionBegin;
356   CHKERR readMesh();
357   CHKERR setupProblem();
358
359   if constexpr (debug) {
360     CHKERR testSideFE();
361     CHKERR testOp();
362   }
363
364   CHKERR initialiseFieldVelocity();
365
366   maxPtr = boost::make_shared<double>(0);
367   CHKERR refineMesh(WrapperClassInitalSolution(maxPtr));
368
369   auto simple = mField.getInterface<Simple>();
370   simple->getBitRefLevel() = BitRefLevel().set(skeleton_bit) |
371                              BitRefLevel().set(aggregate_bit);
372   simple->getBitRefLevelMask() = BitRefLevel().set();
373   simple->reSetUp(true);
374
375   CHKERR solveAdvection();
376
377   MoFEMFunctionReturn(0);
378 }
379
380 struct LevelSet::OpRhsDomain : public AssemblyDomainEleOp {
381
382   OpRhsDomain(const std::string field_name,
383               boost::shared_ptr<VectorDouble> l_ptr,
384               boost::shared_ptr<VectorDouble> l_dot_ptr,
385               boost::shared_ptr<MatrixDouble> vel_ptr);
386   MoFEMErrorCode iNtegrate(EntData &data);
387
388 private:
389   boost::shared_ptr<VectorDouble> lPtr;
390   boost::shared_ptr<VectorDouble> lDotPtr;
391   boost::shared_ptr<MatrixDouble> velPtr;
392 };
393
394 struct LevelSet::OpLhsDomain : public AssemblyDomainEleOp {
395   OpLhsDomain(const std::string field_name,
396               boost::shared_ptr<MatrixDouble> vel_ptr);
397   MoFEMErrorCode iNtegrate(EntData &row_data, EntData &col_data);
398
399 private:
400   boost::shared_ptr<MatrixDouble> velPtr;
401 };
402
403 struct LevelSet::OpRhsSkeleton : public BoundaryEleOp {
404
405   OpRhsSkeleton(boost::shared_ptr<SideData> side_data_ptr,
406                 boost::shared_ptr<FaceSideEle> side_fe_ptr);
407   MoFEMErrorCode doWork(int side, EntityType type,
408                         EntitiesFieldData::EntData &data);
409
410 private:
411   boost::shared_ptr<SideData> sideDataPtr;
412   boost::shared_ptr<FaceSideEle>
413       sideFEPtr; ///< pointer to element to get data on edge/face sides
414
415   VectorDouble resSkelton;
416 };
417
418 struct LevelSet::OpLhsSkeleton : public BoundaryEleOp {
419   OpLhsSkeleton(boost::shared_ptr<SideData> side_data_ptr,
420                 boost::shared_ptr<FaceSideEle> side_fe_ptr);
421   MoFEMErrorCode doWork(int side, EntityType type,
422                         EntitiesFieldData::EntData &data);
423
424 private:
425   boost::shared_ptr<SideData> sideDataPtr;
426   boost::shared_ptr<FaceSideEle>
427       sideFEPtr; ///< pointer to element to get data on edge/face sides
428
429   MatrixDouble matSkeleton;
430 };
```

Check consitency of tangent matrix and right hand side.

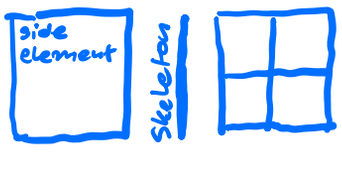
Initial mesh refinment.

Run advection problem

```cpp

int main(int argc, char *argv[]) {

  // Initialisation of MoFEM/PETSc and MOAB data structures
  const char param_file[] = "param_file.petsc";
  MoFEM::Core::Initialize(&argc, &argv, param_file, help);

  try {

    // Create MoAB database
    moab::Core moab_core;
    moab::Interface &moab = moab_core;

    // Create MoFEM database and link it to MoAB
    MoFEM::Core mofem_core(moab);
    MoFEM::Interface &m_field = mofem_core;

    // Register DM Manager
    DMType dm_name = "DMMOFEM";
    CHKERR DMRegister_MoFEM(dm_name);

    // Add logging channel for example
    auto core_log = logging::core::get();
    core_log->add_sink(
        LogManager::createSink(LogManager::getStrmWorld(), "LevelSet"));
    LogManager::setLog("LevelSet");
    MOFEM_LOG_TAG("LevelSet", "LevelSet");

    LevelSet level_set(m_field);
    CHKERR level_set.runProblem();
  }
  CATCH_ERRORS;

  // finish work cleaning memory, getting statistics, etc.
  MoFEM::Core::Finalize();

  return 0;
}

MoFEMErrorCode LevelSet::readMesh() {
  MoFEMFunctionBegin;
  auto simple = mField.getInterface<Simple>();
  // get options from command line
  CHKERR simple->getOptions();

  // Only L2 field is set in this example. Two lines bellow forces simple
  // interface to creat lower dimension (edge) elements, despite that fact that
  // there is no field spanning on such elements. We need them for DG method.
  simple->getAddSkeletonFE() = true;
  simple->getAddBoundaryFE() = true;

  // load mesh file
  simple->getBitRefLevel() = BitRefLevel();
  CHKERR simple->loadFile();

  auto set_problem_bit = [&]() {
    MoFEMFunctionBegin;
    auto bit0 = BitRefLevel().set(start_bit);
    BitRefLevel start_mask;
    for (auto s = 0; s != start_bit; ++s)
      start_mask[s] = true;

    auto bit_mng = mField.getInterface<BitRefManager>();

    Range level0;
    CHKERR bit_mng->getEntitiesByRefLevel(BitRefLevel().set(0),
                                          BitRefLevel().set(), level0);
                                                          , true);
    CHKERR bit_mng->setNthBitRefLevel(level0, current_bit, true);
    CHKERR bit_mng->setNthBitRefLevel(level0, aggregate_bit, true);
    CHKERR bit_mng->setNthBitRefLevel(level0, skeleton_bit, true);

    simple->getBitAdjEnt() = BitRefLevel().set();
    simple->getBitAdjParent() = BitRefLevel().set();
    simple->getBitRefLevel() = BitRefLevel().set(current_bit);
    simple->getBitRefLevelMask() = BitRefLevel().set();

#ifndef NDEBUG
    if constexpr (debug) {
      auto proc_str = boost::lexical_cast<std::string>(mField.get_comm_rank());
      CHKERR bit_mng->writeBitLevelByDim(
          BitRefLevel().set(0), BitRefLevel().set(), SPACE_DIM,
          (proc_str + "level_base.vtk").c_str(), "VTK", "");
    }
#endif
```

if bit reference level is set to zero, simple interface do not set standard bits to reference mesh

Controls adjacencies between DOFs, entities and elements. DOFs of entities of side elements in L^2 space are adjacent to each other.



$p^n \in L^2(\Omega^d)$

```
517     MoFEMFunctionReturn(0);
518   };
519
520   CHKERR set_problem_bit();
521
522   MoFEMFunctionReturn(0);
523 }
524
525 MoFEMErrorCode LevelSet::setupProblem() {
526   MoFEMFunctionBegin;
527   auto simple = mField.getInterface<Simple>();
528   // Scalar fields and vector field is tested. Add more fields, i.e. vector
529   // field if needed.
530   CHKERR simple->addDomainField("L", L2, AINSWORTH_LEGENDRE_BASE, 1);
531   CHKERR simple->addDomainField("V", potential_velocity_space,
532                                 AINSWORTH_LEGENDRE_BASE, 1);
533
534   // set fields order, i.e. for most first cases order is sufficient.
535   CHKERR simple->setFieldOrder("L", 4);
536   CHKERR simple->setFieldOrder("V", 4);
537
538   // setup problem
539   CHKERR simple->setUp();
540
541   MoFEMFunctionReturn(0);
542 }
543
544 FTensor::Index<'i', SPACE_DIM> i;
545 FTensor::Index<'j', SPACE_DIM> j;
546 FTensor::Index<'k', SPACE_DIM> k;
547
548 LevelSet::OpRhsDomain::OpRhsDomain(const std::string field_name,
549                                    boost::shared_ptr<VectorDouble> l_ptr,
550                                    boost::shared_ptr<VectorDouble> l_dot_ptr,
551                                    boost::shared_ptr<MatrixDouble> vel_ptr)
552     : AssemblyDomainEleOp(field_name, field_name, AssemblyDomainEleOp::OPROW),
553       lPtr(l_ptr), lDotPtr(l_dot_ptr), velPtr(vel_ptr) {}
554
555 LevelSet::OpLhsDomain::OpLhsDomain(const std::string field_name,
556                                    boost::shared_ptr<MatrixDouble> vel_ptr)
557     : AssemblyDomainEleOp(field_name, field_name,
558                           AssemblyDomainEleOp::OPROWCOL),
559       velPtr(vel_ptr) {
560   this->sYmm = false;
561 }
562
563 LevelSet::OpRhsSkeleton::OpRhsSkeleton(
564     boost::shared_ptr<SideData> side_data_ptr,
565     boost::shared_ptr<FaceSideEle> side_fe_ptr)
566     : BoundaryEleOp(NOSPACE, BoundaryEleOp::OPSPACE),
567       sideDataPtr(side_data_ptr), sideFEPtr(side_fe_ptr) {}
568
569 LevelSet::OpLhsSkeleton::OpLhsSkeleton(
570     boost::shared_ptr<SideData> side_data_ptr,
571     boost::shared_ptr<FaceSideEle> side_fe_ptr)
572     : BoundaryEleOp(NOSPACE, BoundaryEleOp::OPSPACE),
573       sideDataPtr(side_data_ptr), sideFEPtr(side_fe_ptr) {}
574
575 MoFEMErrorCode LevelSet::OpRhsDomain::iNtegrate(EntData &data) {
576   MoFEMFunctionBegin;
577
578   const auto nb_int_points = getGaussPts().size2();
579   const auto nb_dofs = data.getIndices().size();
580   const auto nb_base_func = data.getN().size2();
581
582   auto t_l = getFTensor0FromVec(*lPtr);
583   auto t_l_dot = getFTensor0FromVec(*lDotPtr);
584   auto t_vel = getFTensor1FromMat<SPACE_DIM>(*velPtr);
585
586   auto t_base = data.getFTensor0N();
587   auto t_diff_base = data.getFTensor1DiffN<SPACE_DIM>();
588
589   auto t_w = getFTensor0IntegrationWeight();
590   for (auto gg = 0; gg != nb_int_points; ++gg) {
591     const auto alpha = t_w * getMeasure();
592     auto res0 = alpha * t_l_dot;
593     FTensor::Tensor1<double, SPACE_DIM> t_res1;
594     t_res1(i) = (alpha * t_l) * t_vel(i);
595     ++t_w;
596     ++t_l;
597     ++t_l_dot;
598     ++t_vel;
599
600     auto &nf = this->locF;
601
602     int rr = 0;
```

set 4th order to all elements to velocity and level set field.

Integrate "rhs" on domain elements.

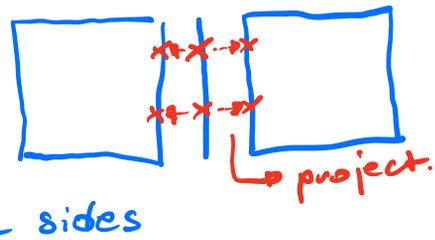
```
603     for (; rr != nb_dofs; ++rr) {
604       nf(rr) += res0 * t_base - t_res1(i) * t_diff_base(i);
605       ++t_base;
606       ++t_diff_base;
607     }
608     for (; rr < nb_base_func; ++rr) {
609       ++t_base;
610       ++t_diff_base;
611     }
612   }
613
614   MoFEMFunctionReturn(0);
615 }
616
617 MoFEMErrorCode LevelSet::OpLhsDomain::iNtegrate(EntData &row_data,
618                                                 EntData &col_data) {
619   MoFEMFunctionBegin;
620
621   const auto nb_int_points = getGaussPts().size2();
622   const auto nb_base_func = row_data.getN().size2();
623   const auto nb_row_dofs = row_data.getIndices().size();
624   const auto nb_col_dofs = col_data.getIndices().size();
625
626   auto t_vel = getFTensor1FromMat<SPACE_DIM>(*velPtr);
627
628   auto t_row_base = row_data.getFTensor0N();
629   auto t_row_diff_base = row_data.getFTensor1DiffN<SPACE_DIM>();
630
631   auto t_w = getFTensor0IntegrationWeight();
632   for (auto gg = 0; gg != nb_int_points; ++gg) {
633     const auto alpha = t_w * getMeasure();
634     const auto beta = alpha * getTSa();
635     ++t_w;
636
637     auto &mat = this->locMat;
638
639     int rr = 0;
640     for (; rr != nb_row_dofs; ++rr) {
641       auto t_col_base = col_data.getFTensor0N(gg, 0);
642       for (int cc = 0; cc != nb_col_dofs; ++cc) {
643         mat(rr, cc) +=
644             (beta * t_row_base - alpha * (t_row_diff_base(i) * t_vel(i))) *
645             t_col_base;
646         ++t_col_base;
647       }
648       ++t_row_base;
649       ++t_row_diff_base;
650     }
651     for (; rr < nb_base_func; ++rr) {
652       ++t_row_base;
653       ++t_row_diff_base;
654     }
655
656     ++t_vel;
657   }
658
659   MoFEMFunctionReturn(0);
660 }
661
662 MoFEMErrorCode
663 LevelSet::OpRhsSkeleton::doWork(int side, EntityType type,
664                                 EntitiesFieldData::EntData &data) {
665   MoFEMFunctionBegin;
666
667   // Collect data from side domain elements
668   CHKERR loopSideFaces("dFE", *sideFEPtr);
669   const auto in_the_loop =
670       sideFEPtr->nInTheLoop; // return number of elements on the side
671
672   auto not_side = [](auto s) {
673     return s == LEFT_SIDE ? RIGHT_SIDE : LEFT_SIDE;
674   };
675
676   auto get_ntensor = [](auto &base_mat) {
677     return FTensor::Tensor0<FTensor::PackPtr<double *, 1>>(
678         &*base_mat.data().begin());
679   };
680
681   if (in_the_loop > 0) {
682
683     // get normal of the face or edge
684     auto t_normal = getFTensor1Normal();
685     const auto nb_gauss_pts = getGaussPts().size2();
686
687     for (auto s0 : {LEFT_SIDE, RIGHT_SIDE}) {
688
```

$$f^a = \sum_i w_i \left(\delta\varphi_a^h(\xi_i)\,\varphi^h(\xi_i) - \frac{\partial \delta\varphi_a^h}{\partial x_i} \cdot v^h(\xi_i) \cdot \varphi^h(\xi_i) \right)$$

Calculate tangent matrix on domain element.

Calculate "rhs" on skeleton.

Iterate side elements & their parents to get data on base functions, indices, and field values. Integration points are projected on elements on the side of skeleton



iterate sides

```
689       // gent number of DOFs on the right side.
690       const auto nb_rows = sideDataPtr->indicesRowSideMap[s0].size();
691
692       if (nb_rows) {
693
694         resSkelton.resize(nb_rows, false);
695         resSkelton.clear();
696
697         // get orientation of the local element edge
698         const auto opposite_s0 = not_side(s0);
699         const auto sense_row = sideDataPtr->senseMap[s0];
700 #ifndef NDEBUG
701         const auto opposite_sense_row = sideDataPtr->senseMap[opposite_s0];
702         if (sense_row * opposite_sense_row > 0)
703           SETERRQ(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
704                   "Should be opposite sign");
705 #endif
706
707         // iterate the side cols
708         const auto nb_row_base_functions =
709             sideDataPtr->rowBaseSideMap[s0].size2();
710
711         auto t_w = getFTensor0IntegrationWeight();
712         auto arr_t_l =
713             make_array(getFTensor0FromVec(sideDataPtr->lVec[LEFT_SIDE]),
714                        getFTensor0FromVec(sideDataPtr->lVec[RIGHT_SIDE]));
715         auto arr_t_vel = make_array(
716             getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[LEFT_SIDE]),
717             getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[RIGHT_SIDE]));
718
719         auto next = [&]() {
720           for (auto &t_l : arr_t_l)
721             ++t_l;
722           for (auto &t_vel : arr_t_vel)
723             ++t_vel;
724         };
725
726 #ifndef NDEBUG
727         if (nb_gauss_pts != sideDataPtr->rowBaseSideMap[s0].size1())
728           SETERRQ(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
729                   "Inconsistent number of DOFs");
730 #endif
731
732         auto t_row_base = get_ntensor(sideDataPtr->rowBaseSideMap[s0]);
733         for (int gg = 0; gg != nb_gauss_pts; ++gg) {
734           FTensor::Tensor1<double, SPACE_DIM> t_vel;
735           t_vel(i) = (arr_t_vel[LEFT_SIDE](i) + arr_t_vel[RIGHT_SIDE](i)) / 2.;
736           const auto dot = sense_row * (t_normal(i) * t_vel(i));
737           const auto l_upwind_side = (dot > 0) ? s0 : opposite_s0;
738           const auto l_upwind = arr_t_l[l_upwind_side];
739           const auto res = t_w * dot * l_upwind;
740           next();
741           ++t_w;
742           auto rr = 0;
743           for (; rr != nb_rows; ++rr) {
744             resSkelton[rr] += t_row_base * res;
745             ++t_row_base;
746           }
747           for (; rr < nb_row_base_functions; ++rr) {
748             ++t_row_base;
749           }
750         }
751         // assemble local operator vector to global vector
752         CHKERR ::VecSetValues(getTSf(),
753                               sideDataPtr->indicesRowSideMap[s0].size(),
754                               &*sideDataPtr->indicesRowSideMap[s0].begin(),
755                               &*resSkelton.begin(), ADD_VALUES);
756       }
757     }
758   }
759
760   MoFEMFunctionReturn(0);
761 }
762
763 MoFEMErrorCode
764 LevelSet::OpLhsSkeleton::doWork(int side, EntityType type,
765                                 EntitiesFieldData::EntData &data) {
766   MoFEMFunctionBegin;
767
768   // Collect data from side domain elements
769   CHKERR loopSideFaces("dFE", *sideFEPtr);
770   const auto in_the_loop =
771       sideFEPtr->nInTheLoop; // return number of elements on the side
772
773   auto not_side = [](auto s) {
774     return s == LEFT_SIDE ? RIGHT_SIDE : LEFT_SIDE;
```

wind

upwind side.

select upwind side

$\sum_i w_i \| S \varphi''(\xi_i) \| \langle v(\xi_i) \cdot n \rangle \varphi_{up}(\xi_i)$

assemble "rhs"

Calculate tangent matrix on skeleton.

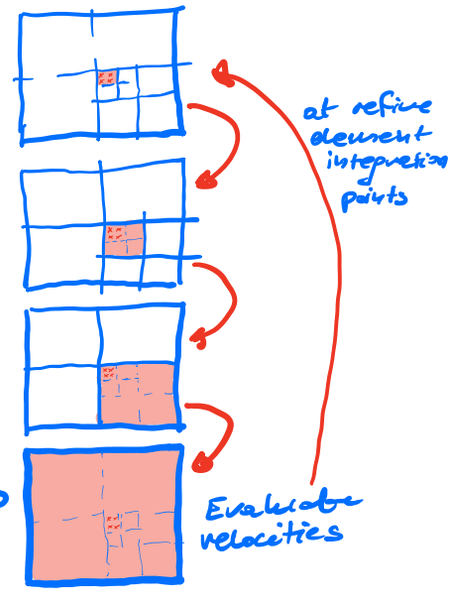
```
775   };
776
777   auto get_ntensor = [](auto &base_mat) {
778     return FTensor::Tensor0<FTensor::PackPtr<double *, 1>>(
779         &*base_mat.data().begin());
780   };
781
782   if (in_the_loop > 0) {
783
784     // get normal of the face or edge
785     auto t_normal = getFTensor1Normal();
786     const auto nb_gauss_pts = getGaussPts().size2();
787
788     for (auto s0 : {LEFT_SIDE, RIGHT_SIDE}) {
789
790       // gent number of DOFs on the right side.
791       const auto nb_rows = sideDataPtr->indicesRowSideMap[s0].size();
792
793       if (nb_rows) {
794
795         // get orientation of the local element edge
796         const auto opposite_s0 = not_side(s0);
797         const auto sense_row = sideDataPtr->senseMap[s0];
798
799         // iterate the side cols
800         const auto nb_row_base_functions =
801             sideDataPtr->rowBaseSideMap[s0].size2();
802
803         for (auto s1 : {LEFT_SIDE, RIGHT_SIDE}) {
804
805           // gent number of DOFs on the right side.
806           const auto nb_cols = sideDataPtr->indicesColSideMap[s1].size();
807           const auto sense_col = sideDataPtr->senseMap[s1];
808
809           // resize local element matrix
810           matSkeleton.resize(nb_rows, nb_cols, false);
811           matSkeleton.clear();
812
813           auto t_w = getFTensor0IntegrationWeight();
814           auto arr_t_vel = make_array(
815               getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[LEFT_SIDE]),
816               getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[RIGHT_SIDE]));
817
818           auto next = [&]() {
819             for (auto &t_vel : arr_t_vel)
820               ++t_vel;
821           };
822
823           auto t_row_base = get_ntensor(sideDataPtr->rowBaseSideMap[s0]);
824           for (int gg = 0; gg != nb_gauss_pts; ++gg) {
825             FTensor::Tensor1<double, SPACE_DIM> t_vel;
826             t_vel(i) =
827                 (arr_t_vel[LEFT_SIDE](i) + arr_t_vel[RIGHT_SIDE](i)) / 2.;
828             const auto dot = sense_row * (t_normal(i) * t_vel(i));
829             const auto l_upwind_side = (dot > 0) ? s0 : opposite_s0;
830             const auto sense_upwind = sideDataPtr->senseMap[l_upwind_side];
831             auto res = t_w * dot; // * sense_row * sense_upwind;
832             next();
833             ++t_w;
834             auto rr = 0;
835             if (s1 == l_upwind_side) {
836               for (; rr != nb_rows; ++rr) {
837                 auto get_ntensor = [](auto &base_mat, auto gg, auto bb) {
838                   double *ptr = &base_mat(gg, bb);
839                   return FTensor::Tensor0<FTensor::PackPtr<double *, 1>>(ptr);
840                 };
841                 auto t_col_base =
842                     get_ntensor(sideDataPtr->colBaseSideMap[s1], gg, 0);
843                 const auto res_row = res * t_row_base;
844                 ++t_row_base;
845                 // iterate columns
846                 for (size_t cc = 0; cc != nb_cols; ++cc) {
847                   matSkeleton(rr, cc) += res_row * t_col_base;
848                   ++t_col_base;
849                 }
850               }
851             }
852             for (; rr < nb_row_base_functions; ++rr) {
853               ++t_row_base;
854             }
855           }
856           // assemble system
857           CHKERR ::MatSetValues(getTSB(),
858                                 sideDataPtr->indicesRowSideMap[s0].size(),
859                                 &*sideDataPtr->indicesRowSideMap[s0].begin(),
860                                 sideDataPtr->indicesColSideMap[s1].size(),
```

```
861                                 &*sideDataPtr->indicesColSideMap[s1].begin(),
862                                 &*matSkeleton.data().begin(), ADD_VALUES);
863         }
864       }
865     }
866   }
867   MoFEMFunctionReturn(0);
868 }
869
870 ForcesAndSourcesCore::UserDataOperator *
871 LevelSet::getZeroLevelVelOp(boost::shared_ptr<MatrixDouble> vel_ptr) {
872   auto get_parent_vel_this = [&]() {
873     auto parent_fe_ptr = boost::make_shared<DomianParentEle>(mField);
874     CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
875         parent_fe_ptr->getOpPtrVector(), {potential_velocity_space});
876     parent_fe_ptr->getOpPtrVector().push_back(
877         new OpCalculateHcurlVectorCurl<potential_velocity_field_dim, SPACE_DIM>(
878             "V", vel_ptr));
879     return parent_fe_ptr;
880   };
881
882   auto get_parents_vel_fe_ptr = [&](auto this_fe_ptr) {
883     std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
884     for (int l = 0; l <= nb_levels; ++l)
885       parents_elems_ptr_vec.emplace_back(
886           boost::make_shared<DomianParentEle>(mField));
887     for (auto l = 1; l <= nb_levels; ++l) {
888       parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
889           new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
890                           BitRefLevel().set(0).flip(), this_fe_ptr,
891                           BitRefLevel().set(0), BitRefLevel().set()));
892     }
893     return parents_elems_ptr_vec[0];
894   };
895
896   auto this_fe_ptr = get_parent_vel_this();
897   auto parent_fe_ptr = get_parents_vel_fe_ptr(this_fe_ptr);
898   return new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
899                          BitRefLevel().set(0).flip(), this_fe_ptr,
900                          BitRefLevel().set(0), BitRefLevel().set());
901 }
902
903 MoFEMErrorCode LevelSet::pushOpDomain() {
904   MoFEMFunctionBegin;
905   auto pip = mField.getInterface<PipelineManager>(); // get interface to
906                                                      // pipeline manager
907
908   pip->getOpDomainLhsPipeline().clear();
909   pip->getOpDomainRhsPipeline().clear();
910
911   pip->setDomainLhsIntegrationRule([](int, int, int o) { return 3 * o; });
912   pip->setDomainRhsIntegrationRule([](int, int, int o) { return 3 * o; });
913
914   pip->getDomainLhsFE()->exeTestHook = [&](FEMethod *fe_ptr) {
915     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
916         current_bit);
917   };
918   pip->getDomainRhsFE()->exeTestHook = [&](FEMethod *fe_ptr) {
919     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
920         current_bit);
921   };
922
923   auto l_ptr = boost::make_shared<VectorDouble>();
924   auto l_dot_ptr = boost::make_shared<VectorDouble>();
925   auto vel_ptr = boost::make_shared<MatrixDouble>();
926
927   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
928       pip->getOpDomainRhsPipeline(), {potential_velocity_space, L2});
929   pip->getOpDomainRhsPipeline().push_back(
930       new OpCalculateScalarFieldValues("L", l_ptr));
931   pip->getOpDomainRhsPipeline().push_back(
932       new OpCalculateScalarFieldValuesDot("L", l_dot_ptr));
933   pip->getOpDomainRhsPipeline().push_back(getZeroLevelVelOp(vel_ptr));
934   pip->getOpDomainRhsPipeline().push_back(
935       new OpRhsDomain("L", l_ptr, l_dot_ptr, vel_ptr));
936
937   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
938       pip->getOpDomainLhsPipeline(), {potential_velocity_space, L2});
939   pip->getOpDomainLhsPipeline().push_back(getZeroLevelVelOp(vel_ptr));
940   pip->getOpDomainLhsPipeline().push_back(new OpLhsDomain("L", vel_ptr));
941
942   MoFEMFunctionReturn(0);
943 }
944
945 boost::shared_ptr<FaceSideEle>
946 LevelSet::getSideFE(boost::shared_ptr<SideData> side_data_ptr) {
```

Get operator calculating velocities on parent element.

calculate curl of velocity potential field

project integration points on parent.



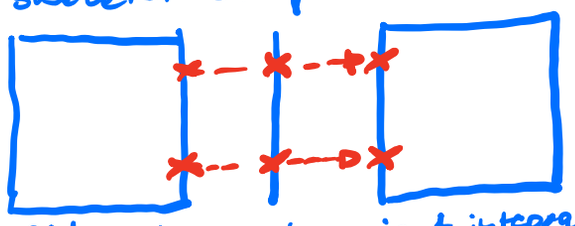
push operators to integrate over domain elements.

Integrate only on elements with current bit reference level on.

Create side element collecting data from elements on the side of skeleton

```
947
948   auto simple = mField.getInterface<Simple>();
949
950   auto l_ptr = boost::make_shared<VectorDouble>();
951   auto vel_ptr = boost::make_shared<MatrixDouble>();
952
953   struct OpSideData : public FaceSideEleOp {
954     OpSideData(boost::shared_ptr<SideData> side_data_ptr)
955         : FaceSideEleOp("L", "L", FaceSideEleOp::OPROWCOL),
956           sideDataPtr(side_data_ptr) {
957       std::fill(&doEntities[MBVERTEX], &doEntities[MBMAXTYPE], false);
958       for (auto t = moab::CN::TypeDimensionMap[SPACE_DIM].first;
959            t <= moab::CN::TypeDimensionMap[SPACE_DIM].second; ++t)
960         doEntities[t] = true;
961       sYmm = false;
962     }
963
964     MoFEMErrorCode doWork(int row_side, int col_side, EntityType row_type,
965                           EntityType col_type, EntData &row_data,
966                           EntData &col_data) {
967       MoFEMFunctionBegin;
968       if ((CN::Dimension(row_type) == SPACE_DIM) &&
969           (CN::Dimension(col_type) == SPACE_DIM)) {
970
971         auto reset = [&](auto nb_in_loop) {
972           sideDataPtr->feSideHandle[nb_in_loop] = 0;
973           sideDataPtr->indicesRowSideMap[nb_in_loop].clear();
974           sideDataPtr->indicesColSideMap[nb_in_loop].clear();
975           sideDataPtr->rowBaseSideMap[nb_in_loop].clear();
976           sideDataPtr->colBaseSideMap[nb_in_loop].clear();
977           sideDataPtr->senseMap[nb_in_loop] = 0;
978         };
979
980         const auto nb_in_loop = getFEMethod()->nInTheLoop;
981         if (nb_in_loop == 0)
982           for (auto s : {0, 1})
983             reset(s);
984
985         sideDataPtr->currentFESide = nb_in_loop;
986         sideDataPtr->senseMap[nb_in_loop] = getSkeletonSense();
987
988       } else {
989         SETERRQ(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY, "Should not happen");
990       }
991
992       MoFEMFunctionReturn(0);
993     };
994
995   private:
996     boost::shared_ptr<SideData> sideDataPtr;
997   };
998
999   struct OpSideDataOnParent : public DomainEleOp {
1000
1001    OpSideDataOnParent(boost::shared_ptr<SideData> side_data_ptr,
1002                       boost::shared_ptr<VectorDouble> l_ptr,
1003                       boost::shared_ptr<MatrixDouble> vel_ptr)
1004        : DomainEleOp("L", "L", DomainEleOp::OPROWCOL),
1005          sideDataPtr(side_data_ptr), lPtr(l_ptr), velPtr(vel_ptr) {
1006      std::fill(&doEntities[MBVERTEX], &doEntities[MBMAXTYPE], false);
1007      for (auto t = moab::CN::TypeDimensionMap[SPACE_DIM].first;
1008           t <= moab::CN::TypeDimensionMap[SPACE_DIM].second; ++t)
1009        doEntities[t] = true;
1010      sYmm = false;
1011    }
1012
1013    MoFEMErrorCode doWork(int row_side, int col_side, EntityType row_type,
1014                          EntityType col_type, EntData &row_data,
1015                          EntData &col_data) {
1016      MoFEMFunctionBegin;
1017
1018      if ((CN::Dimension(row_type) == SPACE_DIM) &&
1019          (CN::Dimension(col_type) == SPACE_DIM)) {
1020        const auto nb_in_loop = sideDataPtr->currentFESide;
1021        sideDataPtr->feSideHandle[nb_in_loop] = getFEEntityHandle();
1022        sideDataPtr->indicesRowSideMap[nb_in_loop] = row_data.getIndices();
1023        sideDataPtr->indicesColSideMap[nb_in_loop] = col_data.getIndices();
1024        sideDataPtr->rowBaseSideMap[nb_in_loop] = row_data.getN();
1025        sideDataPtr->colBaseSideMap[nb_in_loop] = col_data.getN();
1026        (sideDataPtr->lVec)[nb_in_loop] = *lPtr;
1027        (sideDataPtr->velMat)[nb_in_loop] = *velPtr;
1028
1029 #ifndef NDEBUG
1030        if ((sideDataPtr->lVec)[nb_in_loop].size() !=
1031            (sideDataPtr->velMat)[nb_in_loop].size2())
1032          SETERRQ2(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
```

Side element is called on skeleton entity.



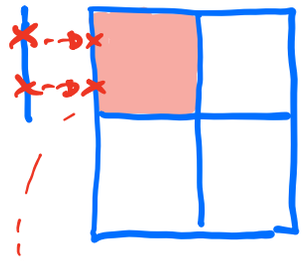
Side element project integration points from skeleton to side.

Clear structure storing date from element on skeleton side.

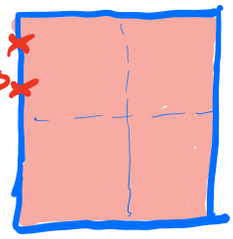
Collect data on skeleton side.

```
1033                  "Wrong number of integaration pts %d != %d",
1034                  (sideDataPtr->lVec)[nb_in_loop].size(),
1035                  (sideDataPtr->velMat)[nb_in_loop].size2());
1036        if ((sideDataPtr->velMat)[nb_in_loop].size1() != SPACE_DIM)
1037          SETERRQ1(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
1038                   "Wrong size of velocity vector size = %d",
1039                   (sideDataPtr->velMat)[nb_in_loop].size1());
1040 #endif
1041
1042        if (!nb_in_loop) {
1043          (sideDataPtr->lVec)[1] = sideDataPtr->lVec[0];
1044          (sideDataPtr->velMat)[1] = (sideDataPtr->velMat)[0];
1045        } else {
1046 #ifndef NDEBUG
1047          if (sideDataPtr->rowBaseSideMap[0].size1() !=
1048              sideDataPtr->rowBaseSideMap[1].size1()) {
1049            SETERRQ2(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
1050                     "Wrong number of integration pt %d != %d",
1051                     sideDataPtr->rowBaseSideMap[0].size1(),
1052                     sideDataPtr->rowBaseSideMap[1].size1());
1053          }
1054          if (sideDataPtr->colBaseSideMap[0].size1() !=
1055              sideDataPtr->colBaseSideMap[1].size1()) {
1056            SETERRQ(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
1057                    "Wrong number of integration pt");
1058          }
1059 #endif
1060        }
1061
1062      } else {
1063        SETERRQ(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY, "Should not happen");
1064      }
1065
1066      MoFEMFunctionReturn(0);
1067    };
1068
1069  private:
1070    boost::shared_ptr<SideData> sideDataPtr;
1071    boost::shared_ptr<VectorDouble> lPtr;
1072    boost::shared_ptr<MatrixDouble> velPtr;
1073  };
1074
1075  // Calculate fields on param mesh bit element
1076  auto get_parent_this = [&]() {
1077    auto parent_fe_ptr = boost::make_shared<DomianParentEle>(mField);
1078    CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1079        parent_fe_ptr->getOpPtrVector(), {potential_velocity_space, L2});
1080    parent_fe_ptr->getOpPtrVector().push_back(
1081        new OpCalculateScalarFieldValues("L", l_ptr));
1082    parent_fe_ptr->getOpPtrVector().push_back(
1083        new OpSideDataOnParent(side_data_ptr, l_ptr, vel_ptr));
1084    return parent_fe_ptr;
1085  };
1086
1087  auto get_parents_fe_ptr = [&](auto this_fe_ptr) {
1088    std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
1089    for (int l = 0; l <= nb_levels; ++l)
1090      parents_elems_ptr_vec.emplace_back(
1091          boost::make_shared<DomianParentEle>(mField));
1092    for (auto l = 1; l <= nb_levels; ++l) {
1093      parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
1094          new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
1095                          BitRefLevel().set(current_bit).flip(), this_fe_ptr,
1096                          BitRefLevel().set(current_bit), BitRefLevel().set()));
1097    }
1098    return parents_elems_ptr_vec[0];
1099  };
1100
1101  // Create aliased shared pointers, all elements are destroyed if side_fe_ptr
1102  // is destroyed
1103  auto get_side_fe_ptr = [&]() {
1104    auto side_fe_ptr = boost::make_shared<FaceSideEle>(mField);
1105
1106    auto this_fe_ptr = get_parent_this();
1107    auto parent_fe_ptr = get_parents_fe_ptr(this_fe_ptr);
1108
1109    side_fe_ptr->getOpPtrVector().push_back(new OpSideData(side_data_ptr));
1110    side_fe_ptr->getOpPtrVector().push_back(getZeroLevelVelOp(vel_ptr));
1111    side_fe_ptr->getOpPtrVector().push_back(
1112        new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
1113                        BitRefLevel().set(current_bit).flip(), this_fe_ptr,
1114                        BitRefLevel().set(current_bit), BitRefLevel().set()));
1115
1116    return side_fe_ptr;
1117  };
1118
```

project/imprint skeleton face on domain interior



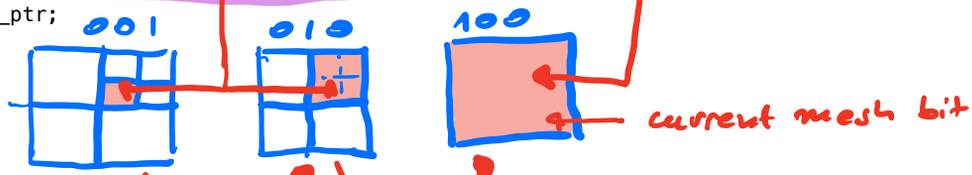
project from domain interior to its parent.



Calculate velocities and level set values at integration points.

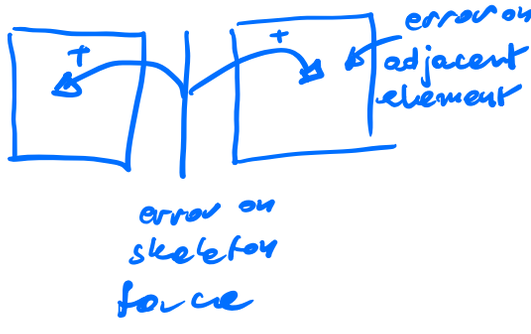
Traverse parents until element with current bit is set.

Create side element and push operators.



```
1119    return get_side_fe_ptr();
1120  };
1121
1122  MoFEMErrorCode LevelSet::pushOpSkeleton() {
1123    MoFEMFunctionBegin;
1124    auto pip = mField.getInterface<PipelineManager>(); // get interface to
1125
1126    pip->getOpSkeletonLhsPipeline().clear();
1127    pip->getOpSkeletonRhsPipeline().clear();
1128
1129    pip->setSkeletonLhsIntegrationRule([](int, int, int o) { return 18; });
1130    pip->setSkeletonRhsIntegrationRule([](int, int, int o) { return 18; });
1131
1132    pip->getSkeletonLhsFE()->exeTestHook = [&](FEMethod *fe_ptr) {
1133      return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1134          skeleton_bit);
1135    };
1136    pip->getSkeletonRhsFE()->exeTestHook = [&](FEMethod *fe_ptr) {
1137      return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1138          skeleton_bit);
1139    };
1140
1141    auto side_data_ptr = boost::make_shared<SideData>();
1142    auto side_fe_ptr = getSideFE(side_data_ptr);
1143
1144    pip->getOpSkeletonRhsPipeline().push_back(
1145        new OpRhsSkeleton(side_data_ptr, side_fe_ptr));
1146    pip->getOpSkeletonLhsPipeline().push_back(
1147        new OpLhsSkeleton(side_data_ptr, side_fe_ptr));
1148
1149    MoFEMFunctionReturn(0);
1150  }
1151
1152  std::tuple<double, Tag> LevelSet::evaluateError() {
1153
1154    struct OpErrorSkel : BoundaryEleOp {
1155
1156      OpErrorSkel(boost::shared_ptr<FaceSideEle> side_fe_ptr,
1157                  boost::shared_ptr<SideData> side_data_ptr,
1158                  SmartPetscObj<Vec> error_sum_ptr, Tag th_error)
1159          : BoundaryEleOp(NOSPACE, BoundaryEleOp::OPSPACE),
1160            sideFEPtr(side_fe_ptr), sideDataPtr(side_data_ptr),
1161            errorSumPtr(error_sum_ptr), thError(th_error) {}
1162
1163      MoFEMErrorCode doWork(int side, EntityType type, EntData &data) {
1164        MoFEMFunctionBegin;
1165
1166        // Collect data from side domain elements
1167        CHKERR loopSideFaces("dFE", *sideFEPtr);
1168        const auto in_the_loop =
1169            sideFEPtr->nInTheLoop; // return number of elements on the side
1170
1171        auto not_side = [](auto s) {
1172          return s == LEFT_SIDE ? RIGHT_SIDE : LEFT_SIDE;
1173        };
1174
1175        auto nb_gauss_pts = getGaussPts().size2();
1176
1177        for (auto s : {LEFT_SIDE, RIGHT_SIDE}) {
1178
1179          auto arr_t_l =
1180              make_array(getFTensor0FromVec(sideDataPtr->lVec[LEFT_SIDE]),
1181                         getFTensor0FromVec(sideDataPtr->lVec[RIGHT_SIDE]));
1182          auto arr_t_vel = make_array(
1183              getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[LEFT_SIDE]),
1184              getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[RIGHT_SIDE]));
1185
1186          auto next = [&]() {
1187            for (auto &t_l : arr_t_l)
1188              ++t_l;
1189            for (auto &t_vel : arr_t_vel)
1190              ++t_vel;
1191          };
1192
1193          double e = 0;
1194          auto t_w = getFTensor0IntegrationWeight();
1195          for (int gg = 0; gg != nb_gauss_pts; ++gg) {
1196            e += t_w * getMeasure() *
1197                 pow(arr_t_l[LEFT_SIDE] - arr_t_l[RIGHT_SIDE], 2);
1198            next();
1199            ++t_w;
1200          }
1201          e = std::sqrt(e);
1202
1203          moab::Interface &moab =
1204              getNumeredEntFiniteElementPtr()->getBasicDataPtr()->moab;
```

Evaluate error by integrating jums on skeleton, and adding error to adjacent domain elements

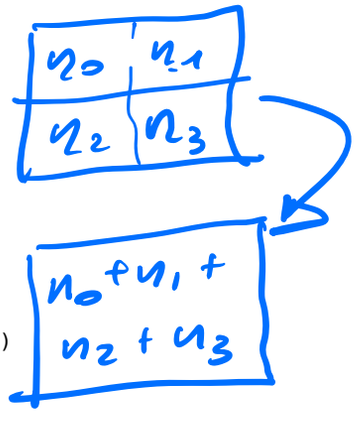


↳ calculate jump on skeleton

```
1205        const void *tags_ptr[2];
1206        CHKERR moab.tag_get_by_ptr(thError, sideDataPtr->feSideHandle.data(), 2,
1207                                   tags_ptr);
1208        for (auto ff : {0, 1}) {
1209          *((double *)tags_ptr[ff]) += e;
1210        }
1211        CHKERR VecSetValue(errorSumPtr, 0, e, ADD_VALUES);
1212      };
1213
1214      MoFEMFunctionReturn(0);
1215    }
1216
1217  private:
1218    boost::shared_ptr<FaceSideEle> sideFEPtr;
1219    boost::shared_ptr<SideData> sideDataPtr;
1220    SmartPetscObj<Vec> errorSumPtr;
1221    Tag thError;
1222  };
1223
1224  auto simple = mField.getInterface<Simple>();
1225
1226  auto error_sum_ptr = createSmartVectorMPI(mField.get_comm(), PETSC_DECIDE, 1);
1227  Tag th_error;
1228  double def_val = 0;
1229  CHKERR mField.get_moab().tag_get_handle("Error", 1, MB_TYPE_DOUBLE, th_error,
1230                                          MB_TAG_CREAT | MB_TAG_SPARSE,
1231                                          &def_val);
1232
1233  auto clear_tags = [&]() {
1234    MoFEMFunctionBegin;
1235    Range fe_ents;
1236    CHKERR mField.get_moab().get_entities_by_dimension(0, SPACE_DIM, fe_ents);
1237    double zero;
1238    CHKERR mField.get_moab().tag_clear_data(th_error, fe_ents, &zero);
1239    MoFEMFunctionReturn(0);
1240  };
1241
1242  auto evaluate_error = [&]() {
1243    MoFEMFunctionBegin;
1244    auto skel_fe = boost::make_shared<BoundaryEle>(mField);
1245    skel_fe->getRuleHook = [](int, int, int o) { return 3 * o; };
1246    auto side_data_ptr = boost::make_shared<SideData>();
1247    auto side_fe_ptr = getSideFE(side_data_ptr);
1248    skel_fe->getOpPtrVector().push_back(
1249        new OpErrorSkel(side_fe_ptr, side_data_ptr, error_sum_ptr, th_error));
1250    auto simple = mField.getInterface<Simple>();
1251
1252    skel_fe->exeTestHook = [&](FEMethod *fe_ptr) {
1253      return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1254          skeleton_bit);
1255    };
1256
1257    CHKERR DMoFEMLoopFiniteElements(simple->getDM(),
1258                                    simple->getSkeletonFEName(), skel_fe);
1259
1260    MoFEMFunctionReturn(0);
1261  };
1262
1263  auto assemble_and_sum = [](auto vec) {
1264    CHK_THROW_MESSAGE(VecAssemblyBegin(vec), "assemble");
1265    CHK_THROW_MESSAGE(VecAssemblyEnd(vec), "assemble");
1266    double sum;
1267    CHK_THROW_MESSAGE(VecSum(vec, &sum), "assemble");
1268    return sum;
1269  };
1270
1271  auto propagate_error_to_parents = [&]() {
1272    MoFEMFunctionBegin;
1273
1274    auto &moab = mField.get_moab();
1275    auto fe_ptr = boost::make_shared<FEMethod>();
1276    fe_ptr->exeTestHook = [&](FEMethod *fe_ptr) {
1277      return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1278          current_bit);
1279    };
1280
1281    fe_ptr->preProcessHook = []() { return 0; };
1282    fe_ptr->postProcessHook = []() { return 0; };
1283    fe_ptr->operatorHook = [&]() {
1284      MoFEMFunctionBegin;
1285
1286      auto fe_ent = fe_ptr->numeredEntFiniteElementPtr->getEnt();
1287      auto parent = fe_ptr->numeredEntFiniteElementPtr->getParentEnt();
1288      auto th_parent = fe_ptr->numeredEntFiniteElementPtr->getBasicDataPtr()
1289                           ->th_RefParentHandle;
1290
```

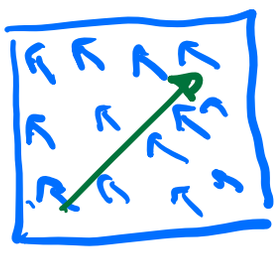
add error to side elements

project error from children to parents.



```
1291      double error;
1292      CHKERR moab.tag_get_data(th_error, &fe_ent, 1, &error);
1293
1294      boost::function<MoFEMErrorCode(EntityHandle, double)> add_error =
1295          [&](auto fe_ent, auto error) {
1296            MoFEMFunctionBegin;
1297            double *e_ptr;
1298            CHKERR moab.tag_get_by_ptr(th_error, &fe_ent, 1,
1299                                       (const void **)&e_ptr);
1300            (*e_ptr) += error;
1301
1302            EntityHandle parent;
1303            CHKERR moab.tag_get_data(th_parent, &fe_ent, 1, &parent);
1304            if (parent != fe_ent && parent)
1305              CHKERR add_error(parent, *e_ptr);
1306
1307            MoFEMFunctionReturn(0);
1308          };
1309
1310      CHKERR add_error(parent, error);
1311
1312      MoFEMFunctionReturn(0);
1313    };
1314
1315    CHKERR DMoFEMLoopFiniteElements(simple->getDM(), simple->getDomainFEName(),
1316                                    fe_ptr);
1317
1318    MoFEMFunctionReturn(0);
1319  };
1320
1321  CHK_THROW_MESSAGE(clear_tags(), "clear error tags");
1322  CHK_THROW_MESSAGE(evaluate_error(), "evaluate error");
1323  CHK_THROW_MESSAGE(propagate_error_to_parents(), "propagate error");
1324
1325  return std::make_tuple(assemble_and_sum(error_sum_ptr), th_error);
1326 }
1327
1328 /**
1329  * @brief test side element
1330  *
1331  * Check consistency between volume and skeleton integral
1332  *
1333  * @return MoFEMErrorCode
1334  */
1335 MoFEMErrorCode LevelSet::testSideFE() {
1336   MoFEMFunctionBegin;
1337
1338   /**
1339    * @brief calculate volume
1340    *
1341    */
1342   struct DivergenceVol : public DomainEleOp {
1343     DivergenceVol(boost::shared_ptr<VectorDouble> l_ptr,
1344                   boost::shared_ptr<MatrixDouble> vel_ptr,
1345                   SmartPetscObj<Vec> div_vec)
1346         : DomainEleOp("L", DomainEleOp::OPROW), lPtr(l_ptr), velPtr(vel_ptr),
1347           divVec(div_vec) {}
1348     MoFEMErrorCode doWork(int side, EntityType type,
1349                           EntitiesFieldData::EntData &data) {
1350       MoFEMFunctionBegin;
1351       const auto nb_dofs = data.getIndices().size();
1352       if (nb_dofs) {
1353         const auto nb_gauss_pts = getGaussPts().size2();
1354         const auto t_w = getFTensor0IntegrationWeight();
1355         auto t_diff = data.getFTensor1DiffN<SPACE_DIM>();
1356         auto t_l = getFTensor0FromVec(*lPtr);
1357         auto t_vel = getFTensor1FromMat<SPACE_DIM>(*velPtr);
1358         double div = 0;
1359         for (auto gg = 0; gg != nb_gauss_pts; ++gg) {
1360           for (int rr = 0; rr != nb_dofs; ++rr) {
1361             div += getMeasure() * t_w * t_l * (t_diff(i) * t_vel(i));
1362             ++t_diff;
1363           }
1364           ++t_w;
1365           ++t_l;
1366           ++t_vel;
1367         }
1368         CHKERR VecSetValue(divVec, 0, div, ADD_VALUES);
1369       }
1370       MoFEMFunctionReturn(0);
1371     }
1372
1373   private:
1374     boost::shared_ptr<VectorDouble> lPtr;
1375     boost::shared_ptr<MatrixDouble> velPtr;
1376     SmartPetscObj<Vec> divVec;
```

Test consitency between volume and skeleton integral



grad φ

Gradient of level set is orthogonal to velocity field. Then is a case of stationary solution.

```
};

/**
 * @brief calculate skeleton integral
 *
 */
struct DivergenceSkeleton : public BoundaryEleOp {
  DivergenceSkeleton(boost::shared_ptr<SideData> side_data_ptr,
                     boost::shared_ptr<FaceSideEle> side_fe_ptr,
                     SmartPetscObj<Vec> div_vec)
      : BoundaryEleOp(NOSPACE, BoundaryEleOp::OPSPACE),
        sideDataPtr(side_data_ptr), sideFEPtr(side_fe_ptr), divVec(div_vec) {}
  MoFEMErrorCode doWork(int side, EntityType type,
                        EntitiesFieldData::EntData &data) {
    MoFEMFunctionBegin;

    auto get_ntensor = [](auto &base_mat) {
      return FTensor::Tensor0<FTensor::PackPtr<double *, 1>>(
          &*base_mat.data().begin());
    };

    auto not_side = [](auto s) {
      return s == LEFT_SIDE ? RIGHT_SIDE : LEFT_SIDE;
    };

    // Collect data from side domain elements
    CHKERR loopSideFaces("dFE", *sideFEPtr);
    const auto in_the_loop =
        sideFEPtr->nInTheLoop; // return number of elements on the side

    auto t_normal = getFTensor1Normal();
    const auto nb_gauss_pts = getGaussPts().size2();
    for (auto s0 : {LEFT_SIDE, RIGHT_SIDE}) {
      const auto nb_dofs = sideDataPtr->indicesRowSideMap[s0].size();
      if (nb_dofs) {
        auto t_base = get_ntensor(sideDataPtr->rowBaseSideMap[s0]);
        auto nb_row_base_functions = sideDataPtr->rowBaseSideMap[s0].size2();
        auto side_sense = sideDataPtr->senseMap[s0];
        auto opposite_s0 = not_side(s0);

        auto arr_t_l =
            make_array(getFTensor0FromVec(sideDataPtr->lVec[LEFT_SIDE]),
                       getFTensor0FromVec(sideDataPtr->lVec[RIGHT_SIDE]));
        auto arr_t_vel = make_array(
            getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[LEFT_SIDE]),
            getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[RIGHT_SIDE]));

        auto next = [&]() {
          for (auto &t_l : arr_t_l)
            ++t_l;
          for (auto &t_vel : arr_t_vel)
            ++t_vel;
        };

        double div = 0;

        auto t_w = getFTensor0IntegrationWeight();
        for (int gg = 0; gg != nb_gauss_pts; ++gg) {
          FTensor::Tensor1<double, SPACE_DIM> t_vel;
          t_vel(i) =
              (arr_t_vel[LEFT_SIDE](i) + arr_t_vel[RIGHT_SIDE](i)) / 2.;
          const auto dot = (t_normal(i) * t_vel(i)) * side_sense;
          const auto l_upwind_side = (dot > 0) ? s0 : opposite_s0;
          const auto l_upwind =
              arr_t_l[l_upwind_side]; //< assume that field is continues,
                                      // initialisation field has to be smooth
                                      // and exactly approximated by approx
                                      // base
          auto res = t_w * l_upwind * dot;
          ++t_w;
          next();
          int rr = 0;
          for (; rr != nb_dofs; ++rr) {
            div += t_base * res;
            ++t_base;
          }
          for (; rr < nb_row_base_functions; ++rr) {
            ++t_base;
          }
        }
        CHKERR VecSetValue(divVec, 0, div, ADD_VALUES);
      }
      if (!in_the_loop)
        break;
    }
```

```
1463      MoFEMFunctionReturn(0);
1464    }
1465
1466  private:
1467    boost::shared_ptr<SideData> sideDataPtr;
1468    boost::shared_ptr<FaceSideEle> sideFEPtr;
1469    boost::shared_ptr<MatrixDouble> velPtr;
1470    SmartPetscObj<Vec> divVec;
1471  };
1472
1473  auto vol_fe = boost::make_shared<DomainEle>(mField);
1474  auto skel_fe = boost::make_shared<BoundaryEle>(mField);
1475
1476  vol_fe->getRuleHook = [](int, int, int o) { return 3 * o; };
1477  skel_fe->getRuleHook = [](int, int, int o) { return 3 * o; };
1478
1479  auto div_vol_vec = createSmartVectorMPI(mField.get_comm(), PETSC_DECIDE, 1);
1480  auto div_skel_vec = createSmartVectorMPI(mField.get_comm(), PETSC_DECIDE, 1);
1481
1482  auto l_ptr = boost::make_shared<VectorDouble>();
1483  auto vel_ptr = boost::make_shared<MatrixDouble>();
1484  auto side_data_ptr = boost::make_shared<SideData>();
1485  auto side_fe_ptr = getSideFE(side_data_ptr);
1486
1487  CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1488      vol_fe->getOpPtrVector(), {potential_velocity_space, L2});
1489  vol_fe->getOpPtrVector().push_back(
1490      new OpCalculateScalarFieldValues("L", l_ptr));
1491  vol_fe->getOpPtrVector().push_back(getZeroLevelVelOp(vel_ptr));
1492  vol_fe->getOpPtrVector().push_back(
1493      new DivergenceVol(l_ptr, vel_ptr, div_vol_vec));
1494
1495  skel_fe->getOpPtrVector().push_back(
1496      new DivergenceSkeleton(side_data_ptr, side_fe_ptr, div_skel_vec));
1497
1498  auto simple = mField.getInterface<Simple>();
1499  auto dm = simple->getDM();
1500
1501  /**
1502   * Set up problem such that gradient of level set field is orthogonal to
1503   * velocity field. Then volume and skeleton integral should yield the same
1504   * value.
1505   */
1506
1507  CHKERR initialiseFieldVelocity(
1508      [](double x, double y, double) { return x - y; });
1509  CHKERR initialiseFieldLevelSet(
1510      [](double x, double y, double) { return x - y; });
1511
1512  vol_fe->exeTestHook = [&](FEMethod *fe_ptr) {
1513    return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1514        current_bit);
1515  };
1516  skel_fe->exeTestHook = [&](FEMethod *fe_ptr) {
1517    return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1518        skeleton_bit);
1519  };
1520
1521  CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(), vol_fe);
1522  CHKERR DMoFEMLoopFiniteElements(dm, simple->getSkeletonFEName(), skel_fe);
1523  CHKERR DMoFEMLoopFiniteElements(dm, simple->getBoundaryFEName(), skel_fe);
1524
1525  auto assemble_and_sum = [](auto vec) {
1526    CHK_THROW_MESSAGE(VecAssemblyBegin(vec), "assemble");
1527    CHK_THROW_MESSAGE(VecAssemblyEnd(vec), "assemble");
1528    double sum;
1529    CHK_THROW_MESSAGE(VecSum(vec, &sum), "assemble");
1530    return sum;
1531  };
1532
1533  auto div_vol = assemble_and_sum(div_vol_vec);
1534  auto div_skel = assemble_and_sum(div_skel_vec);
1535
1536  auto eps = std::abs((div_vol - div_skel) / (div_vol + div_skel));
1537
1538  MOFEM_LOG("WORLD", Sev::inform) << "Testing divergence volume: " << div_vol;
1539  MOFEM_LOG("WORLD", Sev::inform) << "Testing divergence skeleton: " << div_skel
1540                                  << " relative difference: " << eps;
1541
1542  constexpr double eps_err = 1e-6;
1543  if (eps > eps_err)
1544    SETERRQ(PETSC_COMM_SELF, MOFEM_DATA_INCONSISTENCY,
1545            "No consistency between skeleton integral and volume integral");
1546
1547  MoFEMFunctionReturn(0);
1548 };
```

Generate fields

Calculate integrals for domain, skeleton and boundary.

```
1549
1550 MoFEMErrorCode LevelSet::testOp() {
1551   MoFEMFunctionBegin;
1552
1553   // get operators tester
1554   auto simple = mField.getInterface<Simple>();
1555   auto opt = mField.getInterface<OperatorsTester>(); // get interface to
1556                                                      // OperatorsTester
1557   auto pip = mField.getInterface<PipelineManager>(); // get interface to
1558                                                      // pipeline manager
1559
1560   CHKERR pushOpDomain();
1561   CHKERR pushOpSkeleton();
1562
1563   auto post_proc = [&](auto dm, auto f_res, auto out_name) {
1564     MoFEMFunctionBegin;
1565     auto post_proc_fe =
1566         boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
1567
1568     using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
1569
1570     auto l_vec = boost::make_shared<VectorDouble>();
1571     post_proc_fe->getOpPtrVector().push_back(
1572         new OpCalculateScalarFieldValues("L", l_vec, f_res));
1573
1574     post_proc_fe->getOpPtrVector().push_back(
1575
1576         new OpPPMap(
1577
1578             post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),
1579
1580             {{"L", l_vec}},
1581
1582             {},
1583
1584             {}, {})
1585
1586     );
1587
1588     CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
1589                                     post_proc_fe);
1590     post_proc_fe->writeFile(out_name);
1591     MoFEMFunctionReturn(0);
1592   };
1593
1594   constexpr double eps = 1e-4;
1595
1596   auto x =
1597       opt->setRandomFields(simple->getDM(), {{"L", {-1, 1}}, {"V", {-1, 1}}});
1598   auto dot_x = opt->setRandomFields(simple->getDM(), {{"L", {-1, 1}}});
1599   auto diff_x = opt->setRandomFields(simple->getDM(), {{"L", {-1, 1}}});
1600
1601   auto test_domain_ops = [&](auto fe_name, auto lhs_pipeline,
1602                              auto rhs_pipeline) {
1603     MoFEMFunctionBegin;
1604
1605     auto diff_res = opt->checkCentralFiniteDifference(
1606         simple->getDM(), fe_name, rhs_pipeline, lhs_pipeline, x, dot_x,
1607         SmartPetscObj<Vec>(), diff_x, 0, 1, eps);
1608
1609     if constexpr (debug) {
1610       // Example how to plot direction in direction diff_x. If instead
1611       // directionalCentralFiniteDifference(...) diff_res is used, then error
1612       // on directive is plotted.
1613       CHKERR post_proc(simple->getDM(), diff_res, "tangent_op_error.h5m");
1614     }
1615
1616     // Calculate norm of difference between directive calculated from finite
1617     // difference, and tangent matrix.
1618     double fnorm;
1619     CHKERR VecNorm(diff_res, NORM_2, &fnorm);
1620     MOFEM_LOG_C("LevelSet", Sev::inform,
1621                 "Test consistency of tangent matrix %3.4e", fnorm);
1622
1623     constexpr double err = 1e-9;
1624     if (fnorm > err)
1625       SETERRQ1(PETSC_COMM_WORLD, MOFEM_ATOM_TEST_INVALID,
1626                "Norm of directional derivative too large err = %3.4e", fnorm);
1627
1628     MoFEMFunctionReturn(0);
1629   };
1630
1631   CHKERR test_domain_ops(simple->getDomainFEName(), pip->getDomainLhsFE(),
1632                          pip->getDomainRhsFE());
1633   CHKERR test_domain_ops(simple->getSkeletonFEName(), pip->getSkeletonLhsFE(),
1634                          pip->getSkeletonRhsFE());
```

Test tangent matrix in elements, skeleton and domain.

push operators

used to print inconsistency

Generate random fields

Validate directional directive.

$$\Delta = f(x+\Delta x) - f(x) - k\Delta x \approx \varepsilon$$

```
1635
1636   MoFEMFunctionReturn(0);
1637 };
1638
1639 MoFEMErrorCode LevelSet::initialiseFieldLevelSet(
1640     boost::function<double(double, double, double)> level_fun) {
1641   MoFEMFunctionBegin;
1642
1643   // get operators tester
1644   auto simple = mField.getInterface<Simple>();
1645   auto pip = mField.getInterface<PipelineManager>(); // get interface to
1646                                                      // pipeline manager
1647   auto prb_mng = mField.getInterface<ProblemsManager>();
1648
1649   boost::shared_ptr<FEMethod> lhs_fe = boost::make_shared<DomainEle>(mField);
1650   boost::shared_ptr<FEMethod> rhs_fe = boost::make_shared<DomainEle>(mField);
1651   auto swap_fe = [&]() {
1652     lhs_fe.swap(pip->getDomainLhsFE());
1653     rhs_fe.swap(pip->getDomainRhsFE());
1654   };
1655   swap_fe();
1656
1657   pip->setDomainLhsIntegrationRule([](int, int, int o) { return 3 * o; });
1658   pip->setDomainRhsIntegrationRule([](int, int, int o) { return 3 * o; });
1659
1660   auto sub_dm = createSmartDM(mField.get_comm(), "DMMOFEM");
1661   CHKERR DMMoFEMCreateSubDM(sub_dm, simple->getDM(), "LEVELSET_POJECTION");
1662   CHKERR DMMoFEMSetDestroyProblem(sub_dm, PETSC_TRUE);
1663   CHKERR DMMoFEMSetSquareProblem(sub_dm, PETSC_TRUE);
1664   CHKERR DMMoFEMAddElement(sub_dm, simple->getDomainFEName());
1665   CHKERR DMMoFEMAddSubFieldRow(sub_dm, "L");
1666   CHKERR DMSetUp(sub_dm);
1667
1668   BitRefLevel remove_mask = BitRefLevel().set(current_bit);
1669   remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
1670   CHKERR prb_mng->removeDofsOnEntities("LEVELSET_POJECTION", "L",
1671                                        BitRefLevel().set(), remove_mask);
1672   auto test_bit_ele = [&](FEMethod *fe_ptr) {
1673     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
1674         current_bit);
1675   };
1676   pip->getDomainLhsFE()->exeTestHook = test_bit_ele;
1677   pip->getDomainRhsFE()->exeTestHook = test_bit_ele;
1678
1679   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1680       pip->getOpDomainRhsPipeline(), {potential_velocity_space, L2});
1681   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1682       pip->getOpDomainLhsPipeline(), {potential_velocity_space, L2});
1683   pip->getOpDomainLhsPipeline().push_back(new OpMassLL("L", "L"));
1684   pip->getOpDomainRhsPipeline().push_back(new OpSourceL("L", level_fun));
1685
1686   CHKERR mField.getInterface<FieldBlas>()->setField(0, "L");
1687
1688   auto ksp = pip->createKSP(sub_dm);
1689   CHKERR KSPSetDM(ksp, sub_dm);
1690   CHKERR KSPSetFromOptions(ksp);
1691   CHKERR KSPSetUp(ksp);
1692
1693   auto L = smartCreateDMVector(sub_dm);
1694   auto F = smartVectorDuplicate(L);
1695
1696   CHKERR KSPSolve(ksp, F, L);
1697   CHKERR VecGhostUpdateBegin(L, INSERT_VALUES, SCATTER_FORWARD);
1698   CHKERR VecGhostUpdateEnd(L, INSERT_VALUES, SCATTER_FORWARD);
1699   CHKERR DMoFEMMeshToLocalVector(sub_dm, L, INSERT_VALUES, SCATTER_REVERSE);
1700
1701   auto [error, th_error] = evaluateError();
1702   MOFEM_LOG("LevelSet", Sev::inform) << "Error indicator " << error;
1703 #ifndef NDEBUG
1704   auto fe_meshset =
1705       mField.get_finite_element_meshset(simple->getDomainFEName());
1706   std::vector<Tag> tags{th_error};
1707   CHKERR mField.get_moab().write_file("error.h5m", "MOAB",
1708                                       "PARALLEL=WRITE_PART", &fe_meshset, 1,
1709                                       &*tags.begin(), tags.size());
1710 #endif
1711
1712   auto post_proc = [&](auto dm, auto out_name, auto th_error) {
1713     MoFEMFunctionBegin;
1714     auto post_proc_fe =
1715         boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
1716     post_proc_fe->setTagsToTransfer({th_error});
1717     post_proc_fe->exeTestHook = test_bit_ele;
1718
1719     using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
1720
```

Initialise level set field.

```
1721    auto l_vec = boost::make_shared<VectorDouble>();
1722    auto l_grad_mat = boost::make_shared<MatrixDouble>();
1723    CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1724        post_proc_fe->getOpPtrVector(), {potential_velocity_space, L2});
1725    post_proc_fe->getOpPtrVector().push_back(
1726        new OpCalculateScalarFieldValues("L", l_vec));
1727    post_proc_fe->getOpPtrVector().push_back(
1728        new OpCalculateScalarFieldGradient<SPACE_DIM>("L", l_grad_mat));
1729
1730    post_proc_fe->getOpPtrVector().push_back(
1731
1732        new OpPPMap(
1733
1734            post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),
1735
1736            {{"L", l_vec}},
1737
1738            {{"GradL", l_grad_mat}},
1739
1740            {}, {})
1741
1742    );
1743
1744    CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
1745                                    post_proc_fe);
1746    post_proc_fe->writeFile(out_name);
1747    MoFEMFunctionReturn(0);
1748  };
1749
1750  if constexpr (debug)
1751    CHKERR post_proc(sub_dm, "initial_level_set.h5m", th_error);
1752
1753  swap_fe();
1754
1755  MoFEMFunctionReturn(0);
1756 }
1757
1758 MoFEMErrorCode LevelSet::initialiseFieldVelocity(
1759     boost::function<double(double, double, double)> vel_fun) {
1760   MoFEMFunctionBegin;
1761
1762   // get operators tester
1763   auto simple = mField.getInterface<Simple>();
1764   auto pip = mField.getInterface<PipelineManager>(); // get interface to
1765                                                      // pipeline manager
1766   auto prb_mng = mField.getInterface<ProblemsManager>();
1767
1768   boost::shared_ptr<FEMethod> lhs_fe = boost::make_shared<DomainEle>(mField);
1769   boost::shared_ptr<FEMethod> rhs_fe = boost::make_shared<DomainEle>(mField);
1770   auto swap_fe = [&]() {
1771     lhs_fe.swap(pip->getDomainLhsFE());
1772     rhs_fe.swap(pip->getDomainRhsFE());
1773   };
1774   swap_fe();
1775
1776   pip->setDomainLhsIntegrationRule([](int, int, int o) { return 3 * o; });
1777   pip->setDomainRhsIntegrationRule([](int, int, int o) { return 3 * o; });
1778
1779   auto sub_dm = createSmartDM(mField.get_comm(), "DMMOFEM");
1780   CHKERR DMMoFEMCreateSubDM(sub_dm, simple->getDM(), "VELOCITY_PROJECTION");
1781   CHKERR DMMoFEMSetDestroyProblem(sub_dm, PETSC_TRUE);
1782   CHKERR DMMoFEMSetSquareProblem(sub_dm, PETSC_TRUE);
1783   CHKERR DMMoFEMAddElement(sub_dm, simple->getDomainFEName());
1784   CHKERR DMMoFEMAddSubFieldRow(sub_dm, "V");
1785   CHKERR DMSetUp(sub_dm);
1786
1787   // Velocities are calculated only on corse mesh
1788   BitRefLevel remove_mask = BitRefLevel().set(0);
1789   remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
1790   CHKERR prb_mng->removeDofsOnEntities("VELOCITY_PROJECTION", "V",
1791                                        BitRefLevel().set(), remove_mask);
1792
1793   auto test_bit = [&](FEMethod *fe_ptr) {
1794     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(0);
1795   };
1796   pip->getDomainLhsFE()->exeTestHook = test_bit;
1797   pip->getDomainRhsFE()->exeTestHook = test_bit;
1798
1799   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1800       pip->getOpDomainLhsPipeline(), {potential_velocity_space, L2});
1801   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1802       pip->getOpDomainRhsPipeline(), {potential_velocity_space, L2});
1803
1804   pip->getOpDomainLhsPipeline().push_back(new OpMassVV("V", "V"));
1805   pip->getOpDomainRhsPipeline().push_back(new OpSourceV("V", vel_fun));
1806
```

This solve least square problem to approximate velocity (wind) potential field. Similar problem is solved in Sc-1 tutorial.

```
  auto ksp = pip->createKSP(sub_dm);
  CHKERR KSPSetDM(ksp, sub_dm);
  CHKERR KSPSetFromOptions(ksp);
  CHKERR KSPSetUp(ksp);

  auto L = smartCreateDMVector(sub_dm);
  auto F = smartVectorDuplicate(L);

  CHKERR KSPSolve(ksp, F, L);
  CHKERR VecGhostUpdateBegin(L, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR VecGhostUpdateEnd(L, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR DMoFEMMeshToLocalVector(sub_dm, L, INSERT_VALUES, SCATTER_REVERSE);

  auto post_proc = [&](auto dm, auto out_name) {
    MoFEMFunctionBegin;
    auto post_proc_fe =
        boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
    post_proc_fe->exeTestHook = test_bit;

    CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
        post_proc_fe->getOpPtrVector(), {potential_velocity_space});

    using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;

    if constexpr (SPACE_DIM == 2) {
      auto l_vec = boost::make_shared<VectorDouble>();
      auto potential_vec = boost::make_shared<VectorDouble>();
      auto velocity_mat = boost::make_shared<MatrixDouble>();

      post_proc_fe->getOpPtrVector().push_back(
          new OpCalculateScalarFieldValues("V", potential_vec));
      post_proc_fe->getOpPtrVector().push_back(
          new OpCalculateHcurlVectorCurl<potential_velocity_field_dim,
                                         SPACE_DIM>("V", velocity_mat));

      post_proc_fe->getOpPtrVector().push_back(

          new OpPPMap(

              post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),

              {{"VelocityPotential", potential_vec}},

              {{"Velocity", velocity_mat}},

              {}, {})

      );

    } else {
      SETERRQ(PETSC_COMM_SELF, MOFEM_NOT_IMPLEMENTED,
              "3d case not implemented");
    }

    CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
                                    post_proc_fe);
    post_proc_fe->writeFile(out_name);
    MoFEMFunctionReturn(0);
  };

  if constexpr (debug)
    CHKERR post_proc(sub_dm, "initial_velocity_potential.h5m");

  swap_fe();

  MoFEMFunctionReturn(0);
}

LevelSet *level_set_raw_ptr = nullptr;
MoFEM::TsCtx *ts_ctx;

MoFEMErrorCode LevelSet::solveAdvection() {
  MoFEMFunctionBegin;

  // get operators tester
  auto simple = mField.getInterface<Simple>();
  auto pip = mField.getInterface<PipelineManager>(); // get interface to
  auto prb_mng = mField.getInterface<ProblemsManager>();

  CHKERR pushOpDomain();
  CHKERR pushOpSkeleton();

  auto sub_dm = createSmartDM(mField.get_comm(), "DMMOFEM");
  CHKERR DMMoFEMCreateSubDM(sub_dm, simple->getDM(), "ADVECTION");
  CHKERR DMMoFEMSetDestroyProblem(sub_dm, PETSC_TRUE);
  CHKERR DMMoFEMSetSquareProblem(sub_dm, PETSC_TRUE);
```

Solve advection problem

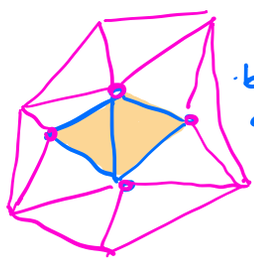
```
  CHKERR DMMoFEMAddElement(sub_dm, simple->getDomainFEName());
  CHKERR DMMoFEMAddElement(sub_dm, simple->getSkeletonFEName());
  CHKERR DMMoFEMAddSubFieldRow(sub_dm, "L");
  CHKERR DMSetUp(sub_dm);

  BitRefLevel remove_mask = BitRefLevel().set(current_bit);
  remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
  CHKERR prb_mng->removeDofsOnEntities("ADVECTION", "L", BitRefLevel().set(),
                                       remove_mask);

  auto add_post_proc_fe = [&]() {
    auto post_proc_fe = boost::make_shared<PostProcEle>(mField);

    Tag th_error;
    double def_val = 0;
    CHKERR mField.get_moab().tag_get_handle(
        "Error", 1, MB_TYPE_DOUBLE, th_error, MB_TAG_CREAT | MB_TAG_SPARSE,
        &def_val);
    post_proc_fe->setTagsToTransfer({th_error});

    post_proc_fe->exeTestHook = [&](FEMethod *fe_ptr) {
      return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
          current_bit);
    };

    using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
    auto l_vec = boost::make_shared<VectorDouble>();
    auto vel_ptr = boost::make_shared<MatrixDouble>();

    CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
        post_proc_fe->getOpPtrVector(), {H1});
    post_proc_fe->getOpPtrVector().push_back(
        new OpCalculateScalarFieldValues("L", l_vec));
    post_proc_fe->getOpPtrVector().push_back(getZeroLevelVelOp(vel_ptr));

    post_proc_fe->getOpPtrVector().push_back(

        new OpPPMap(

            post_proc_fe->getPostProcMesh(),

            post_proc_fe->getMapGaussPts(),

            {{"L", l_vec}},

            {{"V", vel_ptr}},

            {}, {})

    );
    return post_proc_fe;
  };

  auto post_proc_fe = add_post_proc_fe();

  auto set_time_monitor = [&](auto dm, auto ts) {
    auto monitor_ptr = boost::make_shared<FEMethod>();

    monitor_ptr->preProcessHook = []() { return 0; };
    monitor_ptr->operatorHook = []() { return 0; };
    monitor_ptr->postProcessHook = [&]() {
      MoFEMFunctionBegin;

      if (!post_proc_fe)
        SETERRQ(PETSC_COMM_WORLD, MOFEM_DATA_INCONSISTENCY,
                "Null pointer for post proc element");

      CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
                                      post_proc_fe);
      CHKERR post_proc_fe->writeFile(
          "level_set_" +
          boost::lexical_cast<std::string>(monitor_ptr->ts_step) + ".h5m");
      MoFEMFunctionReturn(0);
    };

    boost::shared_ptr<FEMethod> null;
    DMMoFEMTSSetMonitor(sub_dm, ts, simple->getDomainFEName(), monitor_ptr,
                        null, null);

    return monitor_ptr;
  };

  auto ts = pip->createTSIM(sub_dm);

  auto set_solution = [&](auto ts) {
    MoFEMFunctionBegin;
```

```
1979    auto D = smartCreateDMVector(sub_dm);
1980    CHKERR DMoFEMMeshToLocalVector(sub_dm, D, INSERT_VALUES, SCATTER_FORWARD);
1981    CHKERR TSSetSolution(ts, D);
1982    MoFEMFunctionReturn(0);
1983  };
1984  CHKERR set_solution(ts);
1985
1986  auto monitor_pt = set_time_monitor(sub_dm, ts);
1987  CHKERR TSSetFromOptions(ts);
1988
1989  auto B = smartCreateDMMatrix(sub_dm);
1990  CHKERR TSSetIJacobian(ts, B, B, TsSetIJacobian, ts_ctx);
1991  level_set_raw_ptr = this;
1992
1993  CHKERR TSSetUp(ts);
1994
1995  auto ts_pre_step = [](TS ts) {
1996    auto &m_field = level_set_raw_ptr->mField;
1997    auto simple = m_field.getInterface<Simple>();
1998    auto bit_mng = m_field.getInterface<BitRefManager>();
1999    MoFEMFunctionBegin;
2000
2001    auto [error, th_error] = level_set_raw_ptr->evaluateError();
2002    MOFEM_LOG("LevelSet", Sev::inform) << "Error indicator " << error;
2003
2004    auto get_norm = [&](auto x) {
2005      double nrm;
2006      CHKERR VecNorm(x, NORM_2, &nrm);
2007      return nrm;
2008    };
2009
2010    auto set_solution = [&](auto ts) {
2011      MoFEMFunctionBegin;
2012      DM dm;
2013      CHKERR TSGetDM(ts, &dm);
2014      auto prb_ptr = getProblemPtr(dm);
2015
2016      auto x = smartCreateDMVector(dm);
2017      CHKERR DMoFEMMeshToLocalVector(dm, x, INSERT_VALUES, SCATTER_FORWARD);
2018      CHKERR VecGhostUpdateBegin(x, INSERT_VALUES, SCATTER_FORWARD);
2019      CHKERR VecGhostUpdateEnd(x, INSERT_VALUES, SCATTER_FORWARD);
2020
2021      MOFEM_LOG("LevelSet", Sev::inform)
2022          << "Problem " << prb_ptr->getName() << " solution vector norm "
2023          << get_norm(x);
2024      CHKERR TSSetSolution(ts, x);
2025
2026      MoFEMFunctionReturn(0);
2027    };
2028
2029    auto refine_and_project = [&](auto ts) {
2030      MoFEMFunctionBegin;
2031
2032      CHKERR level_set_raw_ptr->refineMesh(
2033          WrapperClassErrorProjection(level_set_raw_ptr->maxPtr));
2034      simple->getBitRefLevel() = BitRefLevel().set(skeleton_bit) |
2035                                 BitRefLevel().set(aggregate_bit) |
2036                                 BitRefLevel().set(aggregate_projection_bit);
2037
2038      simple->reSetUp(true);
2039      DM dm;
2040      CHKERR TSGetDM(ts, &dm);
2041      CHKERR DMSubDMSetUp_MoFEM(dm);
2042
2043      BitRefLevel remove_mask = BitRefLevel().set(current_bit);
2044      remove_mask
2045          .flip(); // DOFs which are not on bit_domain_ele should be removed
2046      CHKERR level_set_raw_ptr->mField.getInterface<ProblemsManager>()
2047          ->removeDofsOnEntities("ADVECTION", "L", BitRefLevel().set(),
2048                                 remove_mask);
2049
2050      MoFEMFunctionReturn(0);
2051    };
2052
2053    auto ts_reset_theta = [&](auto ts) {
2054      MoFEMFunctionBegin;
2055      DM dm;
2056      CHKERR TSGetDM(ts, &dm);
2057
2058      CHKERR TSReset(ts);
2059      CHKERR TSSetUp(ts);
2060
2061      CHKERR level_set_raw_ptr->dgProjection(projection_bit);
2062      CHKERR set_solution(ts);
2063
2064      auto B = smartCreateDMMatrix(dm);
```

1. Calculate error.
2. Project error to parents.
3. Refine mesh.
4. Reset time solver.

```
2065      CHKERR TSSetIJacobian(ts, B, B, TsSetIJacobian, ts_ctx);
2066
2067      MoFEMFunctionReturn(0);
2068    };
2069
2070    CHKERR refine_and_project(ts);
2071    CHKERR ts_reset_theta(ts);
2072
2073    MoFEMFunctionReturn(0);
2074  };
2075
2076  auto ts_post_step = [](TS ts) { return 0; };
2077
2078  CHKERR TSSetPreStep(ts, ts_pre_step);
2079  CHKERR TSSetPostStep(ts, ts_post_step);
2080
2081  CHKERR TSSolve(ts, NULL);
2082
2083  MoFEMFunctionReturn(0);
2084 }
2085
2086 MoFEMErrorCode LevelSet::refineMesh(WrapperClass &&wp) {
2087   MoFEMFunctionBegin;
2088
2089   auto simple = mField.getInterface<Simple>();
2090   auto bit_mng = mField.getInterface<BitRefManager>();
2091   ParallelComm *pcomm =
2092       ParallelComm::get_pcomm(&mField.get_moab(), MYPCOMM_INDEX);
2093   auto proc_str = boost::lexical_cast<std::string>(mField.get_comm_rank());
2094
2095   auto set_bit = [](auto l) { return BitRefLevel().set(l); };
2096
2097   auto save_range = [&](const std::string name, const Range &r) {
2098     MoFEMFunctionBegin;
2099     auto meshset_ptr = get_temp_meshset_ptr(mField.get_moab());
2100     CHKERR mField.get_moab().add_entities(*meshset_ptr, r);
2101     CHKERR mField.get_moab().write_file(name.c_str(), "VTK", "",
2102                                         meshset_ptr->get_ptr(), 1);
2103     MoFEMFunctionReturn(0);
2104   };
2105
2106   // select domain elements to refine by threshold
2107   auto get_refined_elements_meshset = [&](auto bit, auto mask) {
2108     Range fe_ents;
2109     CHKERR bit_mng->getEntitiesByDimAndRefLevel(bit, mask, SPACE_DIM, fe_ents);
2110
2111     Tag th_error;
2112     CHK_MOAB_THROW(mField.get_moab().tag_get_handle("Error", th_error),
2113                    "get error handle");
2114     std::vector<double> errors(fe_ents.size());
2115     CHK_MOAB_THROW(
2116         mField.get_moab().tag_get_data(th_error, fe_ents, &*errors.begin()),
2117         "get tag data");
2118     auto it = std::max_element(errors.begin(), errors.end());
2119     double max;
2120     MPI_Allreduce(&*it, &max, 1, MPI_DOUBLE, MPI_MAX, mField.get_comm());
2121     MOFEM_LOG("LevelSet", Sev::inform) << "Max error: " << max;
2122     auto threshold = wp.getThreshold(max);
2123
2124     std::vector<EntityHandle> fe_to_refine;
2125     fe_to_refine.reserve(fe_ents.size());
2126
2127     auto fe_it = fe_ents.begin();
2128     auto error_it = errors.begin();
2129     for (auto i = 0; i != fe_ents.size(); ++i) {
2130       if (*error_it > threshold) {
2131         fe_to_refine.push_back(*fe_it);
2132       }
2133       ++fe_it;
2134       ++error_it;
2135     }
2136
2137     Range ents;
2138     ents.insert_list(fe_to_refine.begin(), fe_to_refine.end());
2139     CHKERR mField.getInterface<CommInterface>()->synchroniseEntities(ents,
2140                                                                       NOISY);
2141
2142     auto get_neighbours_by_bridge_vertices = [&](auto &&ents) {
2143       Range verts;
2144       CHKERR mField.get_moab().get_connectivity(ents, verts, true);
2145       CHKERR mField.get_moab().get_adjacencies(verts, SPACE_DIM, false, ents,
2146                                                moab::Interface::UNION);
2147       CHKERR mField.getInterface<CommInterface>()->synchroniseEntities(ents);
2148       return ents;
2149     };
2150
```

Select elements for refinement



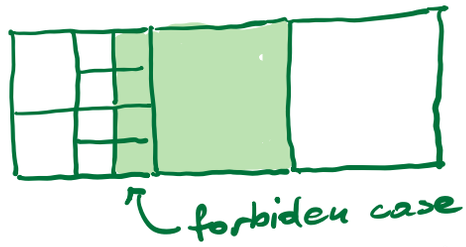
Nodes are bridge adjacencies

```
2151    ents = get_neighbours_by_bridge_vertices(ents);
2152
2153 #ifndef NDEBUG
2154    if (debug) {
2155      auto meshset_ptr = get_temp_meshset_ptr(mField.get_moab());
2156      CHK_MOAB_THROW(mField.get_moab().add_entities(*meshset_ptr, ents),
2157                     "add entities to meshset");
2158      CHKERR mField.get_moab().write_file(
2159          (proc_str + "_fe_to_refine.vtk").c_str(), "VTK", "",
2160          meshset_ptr->get_ptr(), 1);
2161    }
2162 #endif
2163
2164    return ents;
2165  };
2166
2167  // refine elements, and set bit ref level
2168  auto refine_mesh = [&](auto l, auto &&fe_to_refine) {
2169    Skinner skin(&mField.get_moab());
2170    MoFEMFunctionBegin;
2171
2172    // get entities in "l-1" level
2173    Range level_ents;
2174    CHKERR bit_mng->getEntitiesByDimAndRefLevel(
2175        set_bit(start_bit + l - 1), BitRefLevel().set(), SPACE_DIM, level_ents);
2176    // select entities to refine
2177    fe_to_refine = intersect(level_ents, fe_to_refine);
2178    // select entities not to refine
2179    level_ents = subtract(level_ents, fe_to_refine);
2180
2181    // for entities to refine get children, i.e. redlined entities
2182    Range fe_to_refine_children;
2183    bit_mng->updateRangeByChildren(fe_to_refine, fe_to_refine_children);
2184    // add entities to to level "l"
2185    fe_to_refine_children =
2186        fe_to_refine_children.subset_by_dimension(SPACE_DIM);
2187    level_ents.merge(fe_to_refine_children);
2188
2189    auto fix_neighbour_level = [&](auto ll) {
2190      MoFEMFunctionBegin;
2191      auto level_ll = level_ents;
2192      CHKERR bit_mng->filterEntitiesByRefLevel(set_bit(ll), BitRefLevel().set(),
2193                                               level_ll);
2194      Range skin_edges;
2195      CHKERR skin.find_skin(0, level_ll, false, skin_edges);
2196      Range skin_parents;
2197      for (auto lll = 0; lll <= ll; ++lll) {
2198        CHKERR bit_mng->updateRangeByParent(skin_edges, skin_parents);
2199        skin_edges = skin_parents;
2200      }
2201      BitRefLevel bad_bit;
2202      for (auto lll = 0; lll <= ll - 2; ++lll) {
2203        bad_bit[lll] = true;
2204      }
2205      CHKERR bit_mng->filterEntitiesByRefLevel(bad_bit, BitRefLevel().set(),
2206                                               skin_edges);
2207      Range skin_adj_ents;
2208      CHKERR mField.get_moab().get_adjacencies(skin_parents, SPACE_DIM, false,
2209                                               skin_adj_ents,
2210                                               moab::Interface::UNION);
2211      CHKERR bit_mng->filterEntitiesByRefLevel(bad_bit, BitRefLevel().set(),
2212                                               skin_adj_ents);
2213      skin_adj_ents = intersect(skin_adj_ents, level_ents);
2214      if (!skin_adj_ents.empty()) {
2215        level_ents = subtract(level_ents, skin_adj_ents);
2216        Range skin_adj_ents_children;
2217        bit_mng->updateRangeByChildren(skin_adj_ents, skin_adj_ents_children);
2218        level_ents.merge(skin_adj_ents_children);
2219      }
2220      MoFEMFunctionReturn(0);
2221    };
2222
2223    CHKERR fix_neighbour_level(l);
2224
2225    CHKERR mField.getInterface<CommInterface>()->synchroniseEntities(
2226        level_ents);
2227
2228    // get lower dimension entities for level "l"
2229    for (auto d = 0; d != SPACE_DIM; ++d) {
2230      if (d == 0) {
2231        CHKERR mField.get_moab().get_connectivity(
2232            level_ents.subset_by_dimension(SPACE_DIM), level_ents, true);
2233      } else {
2234        CHKERR mField.get_moab().get_adjacencies(
2235            level_ents.subset_by_dimension(SPACE_DIM), d, false, level_ents,
2236            moab::Interface::UNION);
```

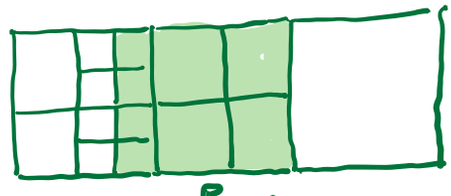
* Get entities on previous refinment

* Substract elements which going be refined Rest of elements will constitute next refined mesh

* Take children of elements to refine and add them to elements of new refined mesh



forbiden case



fix

this code makes fix

```
2237      }
2238    }
2239    CHKERR mField.getInterface<CommInterface>()->synchroniseEntities(
2240        level_ents);
2241
2242    // set bit ref level to level entities
2243    CHKERR bit_mng->setNthBitRefLevel(start_bit + l, false);
2244    CHKERR bit_mng->setNthBitRefLevel(level_ents, start_bit + l, true);
2245
2246 #ifndef NDEBUG
2247    auto proc_str = boost::lexical_cast<std::string>(mField.get_comm_rank());
2248    CHKERR bit_mng->writeBitLevelByDim(
2249        set_bit(start_bit + l), BitRefLevel().set(), SPACE_DIM,
2250        (boost::lexical_cast<std::string>(l) + "_" + proc_str + "_ref_mesh.vtk")
2251            .c_str(),
2252        "VTK", "");
2253 #endif
2254
2255    MoFEMFunctionReturn(0);
2256  };
2257
2258  // set skeleton
2259  auto set_skelton_bit = [&](auto l) {
2260    MoFEMFunctionBegin;
2261
2262    // get entities of dim-1 on level "l"
2263    Range level_edges;
2264    CHKERR bit_mng->getEntitiesByDimAndRefLevel(set_bit(start_bit + l),
2265                                                BitRefLevel().set(),
2266                                                SPACE_DIM - 1, level_edges);
2267
2268    // get parent of entities of level "l"
2269    Range level_edges_parents;
2270    CHKERR bit_mng->updateRangeByParent(level_edges, level_edges_parents);
2271    level_edges_parents =
2272        level_edges_parents.subset_by_dimension(SPACE_DIM - 1);
2273    CHKERR bit_mng->filterEntitiesByRefLevel(
2274        set_bit(start_bit + l), BitRefLevel().set(), level_edges_parents);
2275
2276    // skeleton entities which do not have parents
2277    auto parent_skeleton = intersect(level_edges, level_edges_parents);
2278    auto skeleton = subtract(level_edges, level_edges_parents);
2279
2280    // add adjacent domain entities
2281    CHKERR mField.get_moab().get_adjacencies(unite(parent_skeleton, skeleton),
2282                                             SPACE_DIM, false, skeleton,
2283                                             moab::Interface::UNION);
2284
2285    // set levels
2286    CHKERR mField.getInterface<CommInterface>()->synchroniseEntities(skeleton);
2287    CHKERR bit_mng->setNthBitRefLevel(skeleton_bit, false);
2288    CHKERR bit_mng->setNthBitRefLevel(skeleton, skeleton_bit, true);
2289
2290 #ifndef NDEBUG
2291    CHKERR bit_mng->writeBitLevel(
2292        set_bit(skeleton_bit), BitRefLevel().set(),
2293        (boost::lexical_cast<std::string>(l) + "_" + proc_str + "_skeleton.vtk")
2294            .c_str(),
2295        "VTK", "");
2296 #endif
2297    MoFEMFunctionReturn(0);
2298  };
2299
2300  // Reset bit sand set old current and aggregate bits as projection bits
2301  Range level0_current;
2302  CHKERR bit_mng->getEntitiesByRefLevel(BitRefLevel().set(current_bit),
2303                                        BitRefLevel().set(), level0_current);
2304
2305  Range level0_aggregate;
2306  CHKERR bit_mng->getEntitiesByRefLevel(BitRefLevel().set(aggregate_bit),
2307                                        BitRefLevel().set(), level0_aggregate);
2308
2309  BitRefLevel start_mask;
2310  for (auto s = 0; s != start_bit; ++s)
2311    start_mask[s] = true;
2312  CHKERR bit_mng->lambdaBitRefLevel(
2313      [&](EntityHandle ent, BitRefLevel &bit) { bit &= start_mask; });
2314  CHKERR bit_mng->setNthBitRefLevel(level0_current, projection_bit, true);
2315  CHKERR bit_mng->setNthBitRefLevel(level0_aggregate, aggregate_projection_bit,
2316                                    true);
2317
2318  // Set zero bit ref level
2319  Range level0;
2320  CHKERR bit_mng->getEntitiesByRefLevel(set_bit(0), BitRefLevel().set(),
2321                                        level0);
2322  CHKERR bit_mng->setNthBitRefLevel(level0, start_bit, true);
```

Set bits to mark new refined mesh

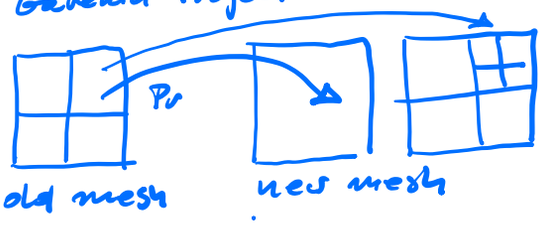
Set bit indicating Skeleton elements, ie. elements on current mesh and their parents.

```
2323   CHKERR bit_mng->setNthBitRefLevel(level0, current_bit, true);
2324   CHKERR bit_mng->setNthBitRefLevel(level0, aggregate_bit, true);
2325   CHKERR bit_mng->setNthBitRefLevel(level0, skeleton_bit, true);
2326
2327   CHKERR wp.setBits(*this, 0);
2328   CHKERR wp.runCalcs(*this, 0);
2329   for (auto l = 0; l != nb_levels; ++l) {
2330     MOFEM_LOG("WORLD", Sev::inform) << "Process level: " << l;
2331     CHKERR refine_mesh(l + 1, get_refined_elements_meshset(
2332                                   set_bit(start_bit + l), BitRefLevel().set()));
2333     CHKERR set_skelton_bit(l + 1);
2334     CHKERR wp.setAggregateBit(*this, l + 1);
2335     CHKERR wp.setBits(*this, l + 1);
2336     CHKERR wp.runCalcs(*this, l + 1);
2337   }
2338
2339   MoFEMFunctionReturn(0);
2340 }
2341
2342 MoFEMErrorCode LevelSet::dgProjection(const int projection_bit) {
2343   MoFEMFunctionBegin;
2344
2345   // get operators tester
2346   auto simple = mField.getInterface<Simple>();
2347   auto pip = mField.getInterface<PipelineManager>(); // get interface to
2348   auto bit_mng = mField.getInterface<BitRefManager>();
2349   auto prb_mng = mField.getInterface<ProblemsManager>();
2350
2351   auto lhs_fe = boost::make_shared<DomainEle>(mField);
2352   auto rhs_fe_prj = boost::make_shared<DomainEle>(mField);
2353   auto rhs_fe_current = boost::make_shared<DomainEle>(mField);
2354
2355   lhs_fe->getRuleHook = [](int, int, int o) { return 3 * o; };
2356   rhs_fe_prj->getRuleHook = [](int, int, int o) { return 3 * o; };
2357   rhs_fe_current->getRuleHook = [](int, int, int o) { return 3 * o; };
2358
2359   auto sub_dm = createSmartDM(mField.get_comm(), "DMMOFEM");
2360   CHKERR DMMoFEMCreateSubDM(sub_dm, simple->getDM(), "DG_PROJECTION");
2361   CHKERR DMMoFEMSetDestroyProblem(sub_dm, PETSC_TRUE);
2362   CHKERR DMMoFEMSetSquareProblem(sub_dm, PETSC_TRUE);
2363   CHKERR DMMoFEMAddElement(sub_dm, simple->getDomainFEName());
2364   CHKERR DMMoFEMAddSubFieldRow(sub_dm, "L");
2365   CHKERR DMSetUp(sub_dm);
2366
2367   Range current_ents;
2368   CHKERR bit_mng->getEntitiesByDimAndRefLevel(BitRefLevel().set(current_bit),
2369                                               BitRefLevel().set(), SPACE_DIM,
2370                                               current_ents);
2371   Range prj_ents;
2372   CHKERR bit_mng->getEntitiesByDimAndRefLevel(BitRefLevel().set(projection_bit),
2373                                               BitRefLevel().set(), SPACE_DIM,
2374                                               prj_ents);
2375   for (auto l = 0; l != nb_levels; ++l) {
2376     CHKERR bit_mng->updateRangeByParent(prj_ents, prj_ents);
2377   }
2378   current_ents = subtract(current_ents, prj_ents);
2379
2380   auto test_mesh_bit = [&](FEMethod *fe_ptr) {
2381     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
2382         current_bit);
2383   };
2384   auto test_prj_bit = [&](FEMethod *fe_ptr) {
2385     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
2386         projection_bit);
2387   };
2388   auto test_current_bit = [&](FEMethod *fe_ptr) {
2389     return current_ents.find(fe_ptr->getFEEntityHandle()) != current_ents.end();
2390   };
2391
2392   lhs_fe->exeTestHook = test_mesh_bit;
2393   rhs_fe_prj->exeTestHook = test_prj_bit;
2394   rhs_fe_current->exeTestHook = test_current_bit;
2395
2396   BitRefLevel remove_mask = BitRefLevel().set(current_bit);
2397   remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
2398   CHKERR prb_mng->removeDofsOnEntities(
2399       "DG_PROJECTION", "L", BitRefLevel().set(), remove_mask, nullptr, 0,
2400       MAX_DOFS_ON_ENTITY, 0, 100, NOISY, true);
2401
2402   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
2403       lhs_fe->getOpPtrVector(), {potential_velocity_space, L2});
2404   lhs_fe->getOpPtrVector().push_back(new OpMassLL("L", "L"));
2405
2406   auto l_vec = boost::make_shared<VectorDouble>();
2407
2408   auto set_prj_from_child = [&](auto rhs_fe_prj) {
```

Run "n" refinment steps.

Project solution from previous refinment & time step, to new mesh after refinment.

This is discontinuous Galerkin Projection



new element is finer (prolongation)
& new element is coarser (restriction)

```
    CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
        rhs_fe_prj->getOpPtrVector(), {potential_velocity_space, L2});
    rhs_fe_prj->getOpPtrVector().push_back(
        new OpCalculateScalarFieldValues("L", l_vec));
    auto get_parent_this = [&]() {
      auto fe_parent_this = boost::make_shared<DomianParentEle>(mField);
      fe_parent_this->getOpPtrVector().push_back(
          new OpScalarFieldL("L", l_vec));
      return fe_parent_this;
    };

    auto get_parents_fe_ptr = [&](auto this_fe_ptr) {
      std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
      for (int l = 0; l <= nb_levels; ++l)
        parents_elems_ptr_vec.emplace_back(
            boost::make_shared<DomianParentEle>(mField));
      for (auto l = 1; l <= nb_levels; ++l) {
        parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
            new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
                            BitRefLevel().set(current_bit).flip(), this_fe_ptr,
                            BitRefLevel().set(current_bit),
                            BitRefLevel().set()));
      }
      return parents_elems_ptr_vec[0];
    };

    auto this_fe_ptr = get_parent_this();
    auto parent_fe_ptr = get_parents_fe_ptr(this_fe_ptr);
    rhs_fe_prj->getOpPtrVector().push_back(
        new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
                        BitRefLevel().set(current_bit).flip(), this_fe_ptr,
                        BitRefLevel().set(current_bit), BitRefLevel().set()));
  };

  auto set_prj_from_parent = [&](auto rhs_fe_current) {
    auto get_parent_this = [&]() {
      auto fe_parent_this = boost::make_shared<DomianParentEle>(mField);
      fe_parent_this->getOpPtrVector().push_back(
          new OpCalculateScalarFieldValues("L", l_vec));
      return fe_parent_this;
    };

    auto get_parents_fe_ptr = [&](auto this_fe_ptr) {
      std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
      for (int l = 0; l <= nb_levels; ++l)
        parents_elems_ptr_vec.emplace_back(
            boost::make_shared<DomianParentEle>(mField));
      for (auto l = 1; l <= nb_levels; ++l) {
        parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
            new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
                            BitRefLevel().set(projection_bit).flip(),
                            this_fe_ptr, BitRefLevel().set(projection_bit),
                            BitRefLevel().set()));
      }
      return parents_elems_ptr_vec[0];
    };

    auto this_fe_ptr = get_parent_this();
    auto parent_fe_ptr = get_parents_fe_ptr(this_fe_ptr);

    auto reset_op_ptr = new DomainEleOp(NOSPACE, DomainEleOp::OPSPACE);
    reset_op_ptr->doWorkRhsHook = [&](DataOperator *op_ptr, int, EntityType,
                                      EntData &) {
      l_vec->resize(static_cast<DomainEleOp *>(op_ptr)->getGaussPts().size2(),
                    false);
      l_vec->clear();
      return 0;
    };
    rhs_fe_current->getOpPtrVector().push_back(reset_op_ptr);
    rhs_fe_current->getOpPtrVector().push_back(
        new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
                        BitRefLevel().set(projection_bit).flip(), this_fe_ptr,
                        BitRefLevel(), BitRefLevel()));
    rhs_fe_current->getOpPtrVector().push_back(new OpScalarFieldL("L", l_vec));
  };

  set_prj_from_child(rhs_fe_prj);
  set_prj_from_parent(rhs_fe_current);

  boost::shared_ptr<FEMethod> null_fe;
  smartGetDMKspCtx(sub_dm)->clearLoops();
  CHKERR DMMoFEMKSPSetComputeOperators(sub_dm, simple->getDomainFEName(),
                                       lhs_fe, null_fe, null_fe);
  CHKERR DMMoFEMKSPSetComputeRHS(sub_dm, simple->getDomainFEName(), rhs_fe_prj,
                                 null_fe, null_fe);
  CHKERR DMMoFEMKSPSetComputeRHS(sub_dm, simple->getDomainFEName(),
```

```
                                rhs_fe_current, null_fe, null_fe);
  auto ksp = MoFEM::createKSP(mField.get_comm());
  CHKERR KSPSetDM(ksp, sub_dm);

  CHKERR KSPSetDM(ksp, sub_dm);
  CHKERR KSPSetFromOptions(ksp);
  CHKERR KSPSetUp(ksp);

  auto L = smartCreateDMVector(sub_dm);
  auto F = smartVectorDuplicate(L);

  CHKERR KSPSolve(ksp, F, L);
  CHKERR VecGhostUpdateBegin(L, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR VecGhostUpdateEnd(L, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR DMoFEMMeshToLocalVector(sub_dm, L, INSERT_VALUES, SCATTER_REVERSE);

  auto [error, th_error] = evaluateError();
  MOFEM_LOG("LevelSet", Sev::inform) << "Error indicator " << error;

  auto post_proc = [&](auto dm, auto out_name, auto th_error) {
    MoFEMFunctionBegin;
    auto post_proc_fe =
        boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
    post_proc_fe->setTagsToTransfer({th_error});
    post_proc_fe->exeTestHook = test_mesh_bit;

    using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;

    auto l_vec = boost::make_shared<VectorDouble>();
    auto l_grad_mat = boost::make_shared<MatrixDouble>();
    CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
        post_proc_fe->getOpPtrVector(), {potential_velocity_space, L2});
    post_proc_fe->getOpPtrVector().push_back(
        new OpCalculateScalarFieldValues("L", l_vec));
    post_proc_fe->getOpPtrVector().push_back(
        new OpCalculateScalarFieldGradient<SPACE_DIM>("L", l_grad_mat));

    post_proc_fe->getOpPtrVector().push_back(

        new OpPPMap(

            post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),

            {{"L", l_vec}},

            {{"GradL", l_grad_mat}},

            {}, {})

    );

    CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
                                    post_proc_fe);
    post_proc_fe->writeFile(out_name);
    MoFEMFunctionReturn(0);
  };

  if constexpr (debug)
    CHKERR post_proc(sub_dm, "dg_projection.h5m", th_error);

  MoFEMFunctionReturn(0);
}
```

Note that field "L" i.e. $\varphi \in L^2(\Omega)$, thus projection can be preformed element by element.
Here we use generic KSP solver and preconditoner. By implementig problem tailored preconditioner using cholesky solver element by element optimisation can be implemented.