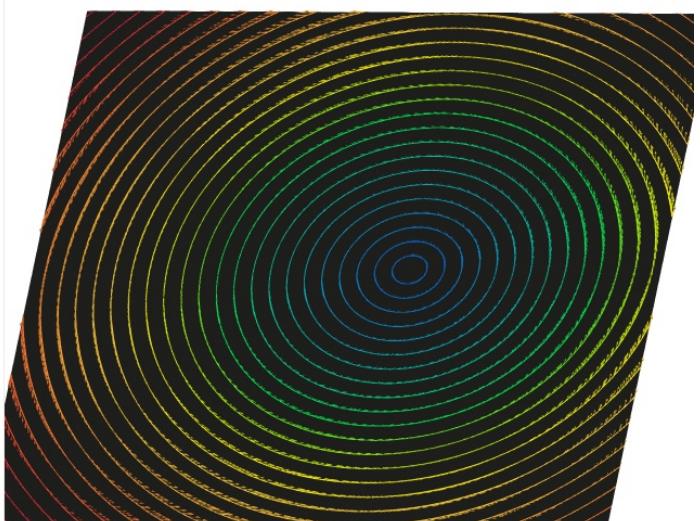
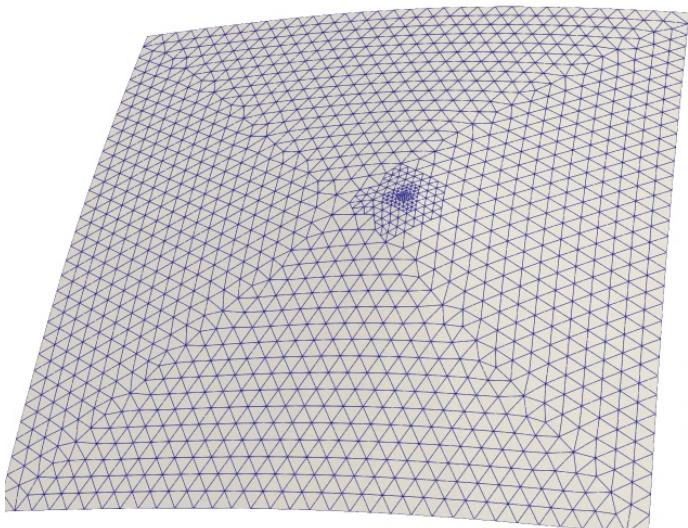


Advection Problem

Transport equation



* Problem setup

* Upwind Discontinuous Galerkin

* Non conforming mesh refinement

* Error indicators and h-adaptivity

* Bit reference levels

* Mass operator

Problem Set Up

Advection problem

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

Velocity potential

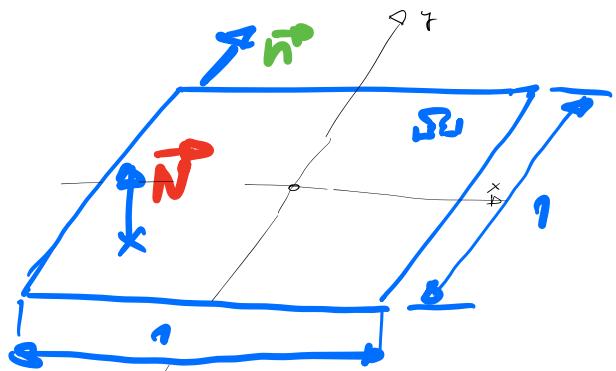
$$\vec{v} = (\vec{\nabla} \times \vec{\phi}), \quad \phi = \Theta \vec{N}$$

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

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

Levi-civita symbol

$$\epsilon_{ij} = \begin{cases} 0 & i=j \\ -1 & i=0 \\ 1 & i=1 \end{cases}$$

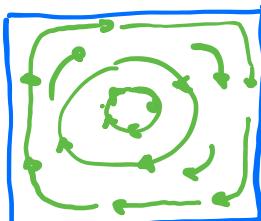


Initial level set

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

$$\Delta x = x - x_c$$

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

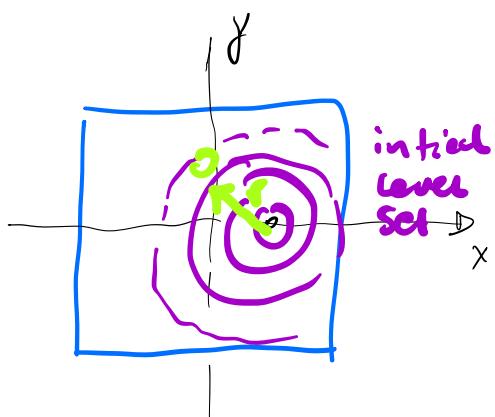


$$v_i u_i = 0$$

zero normal

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

Divergence free



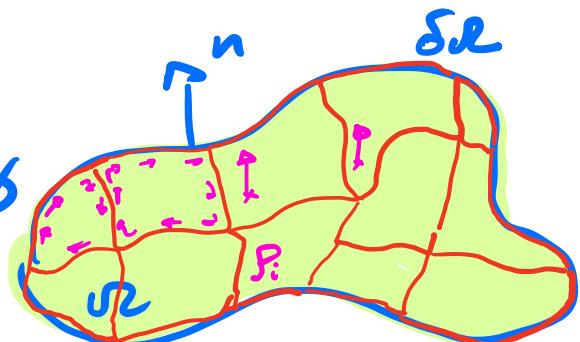
Upwind Discontinuous Galerkin

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

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

$$\int_{\Omega} \delta \varphi \frac{\partial \varphi}{\partial t} dx - \int_{\Omega} \delta \varphi_{,i} v_i \varphi dx$$

$$+ \int_{\partial \Omega} \delta \varphi n_i v_i \varphi d\sigma = 0$$

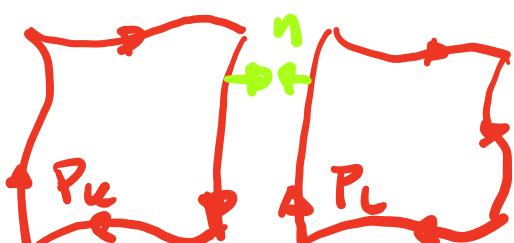


$$\Sigma = \bigcup_k P_k \quad P_k \cap P_l = \emptyset$$

if $k \neq l$

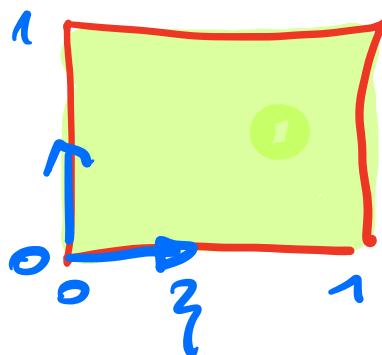
$$\sum_k \int_{P_k} \delta \varphi \frac{\partial \varphi}{\partial t} dx - \sum_k \int_{P_k} \delta \varphi_{,i} v_i \varphi dx$$

$$+ \sum_k \int_{\partial P_k} \delta \varphi \cdot n_i v_i \varphi d\sigma = 0$$



Discretisation in space

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

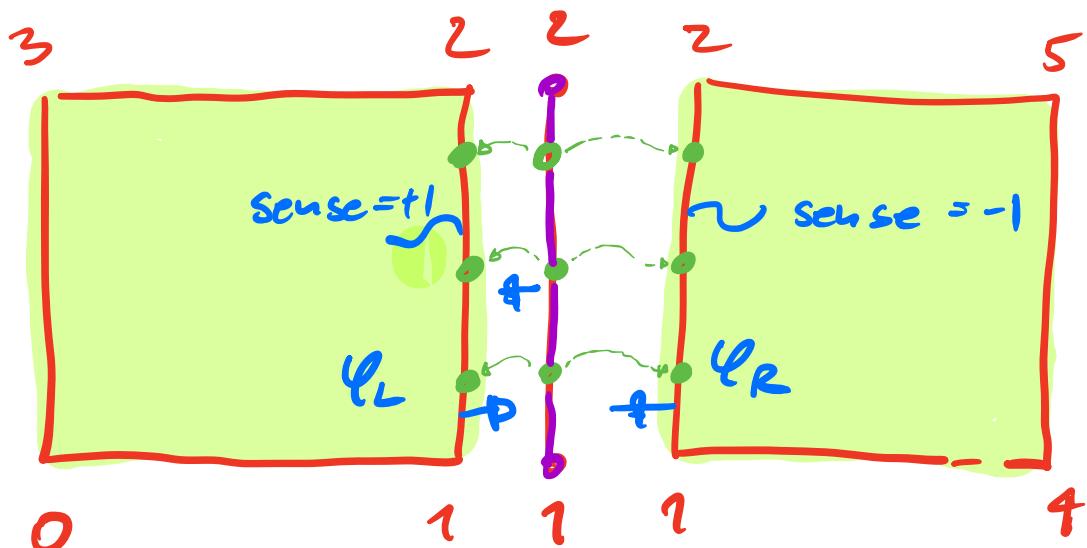


$$\varphi^h(\zeta, \eta) = \sum_k \sum_l L_k(\zeta) L_l(\eta) c_{kl}$$

Legendre Polynomial

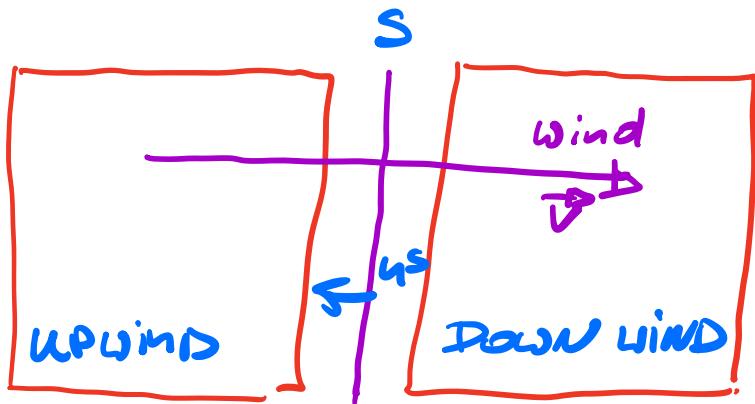
$$\int_0^1 L_u(\zeta) L_v(\zeta) d\zeta = 0$$

orthogonal



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

Upwinding



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

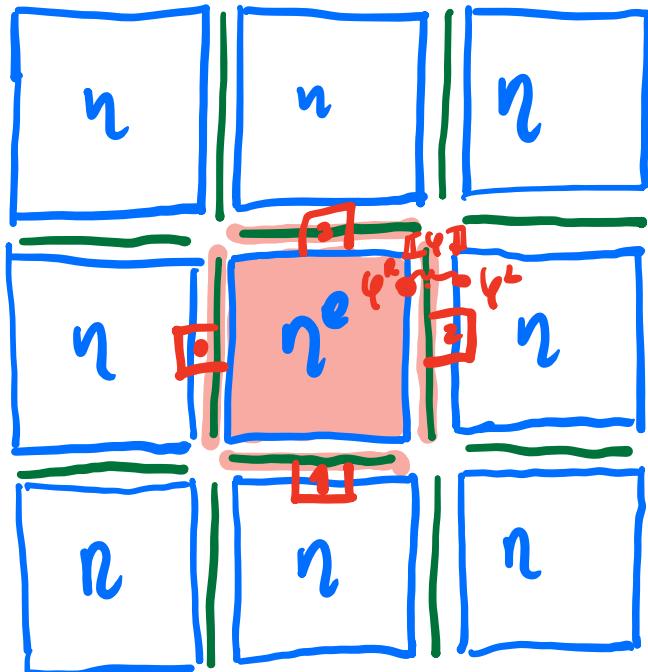
$$\sum_{P_k} \int_{P_k} \left(\delta \varphi^h \frac{\partial \varphi^h}{\partial t} - \delta \varphi_{,i}^h v_i \varphi^h \right) dP_k + \sum_S \int_S [\delta \varphi] \langle v^n \rangle \varphi_{up}^h dS = 0$$

$$[\delta \varphi] = \delta \varphi_e^h - \delta \varphi_L^h$$

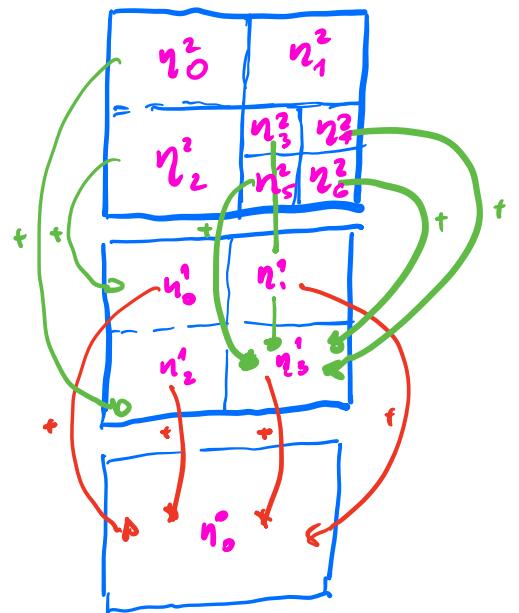
$$\langle v^n \rangle = (v_n^R - v_n^L)/2$$

+ discretisation in time

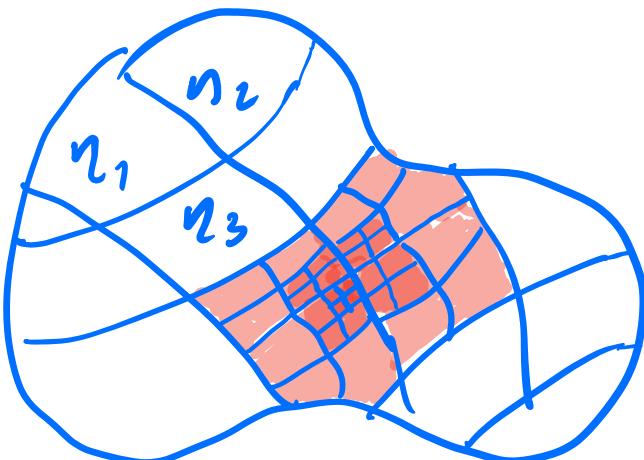
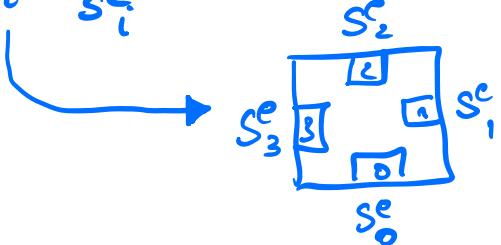
Error indicator



Projection of error from fine to coarse mesh



$$\eta^e = \int_{S^e} \|\varphi\| dS^e = \sum_i \int_{S_i^e} (\varphi^L - \varphi^R) dS_i^e.$$

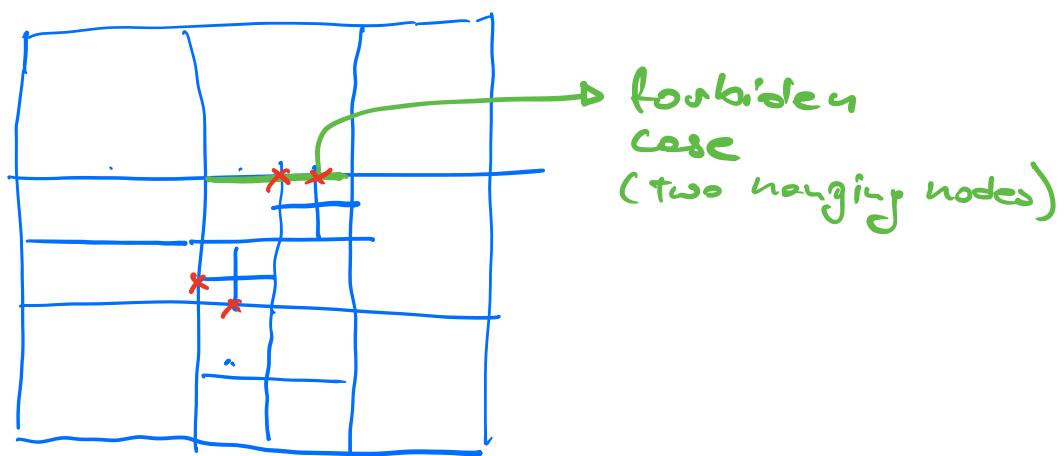


Elements to refine

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

↑
threshold

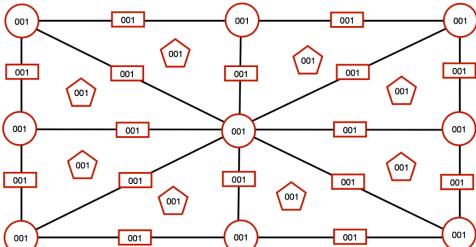
Topological restrictions



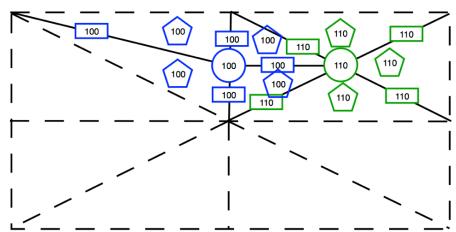
forbidden
case
(two hanging nodes)

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

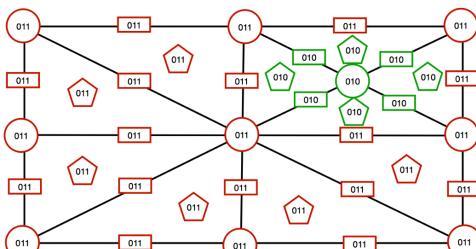
Refined Bit Level 001 and Mask 111



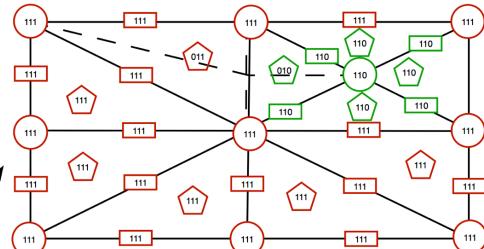
Selection Bit Level 100 and Mask 110



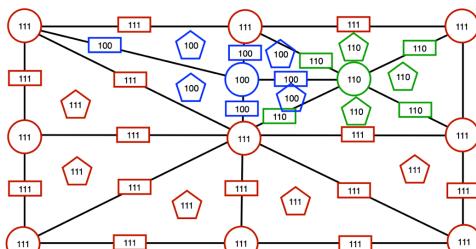
Refined Bit Level 010 and Mask 111



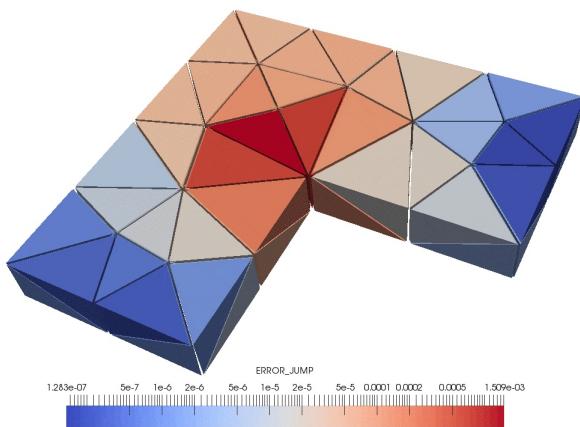
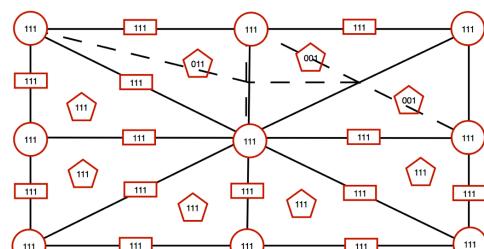
Selection Bit Level 010 and Mask 111



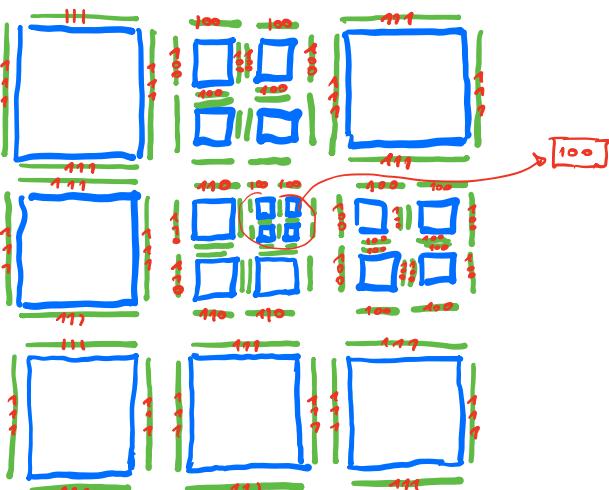
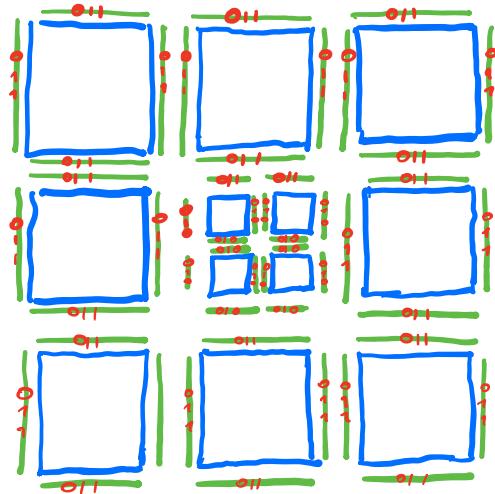
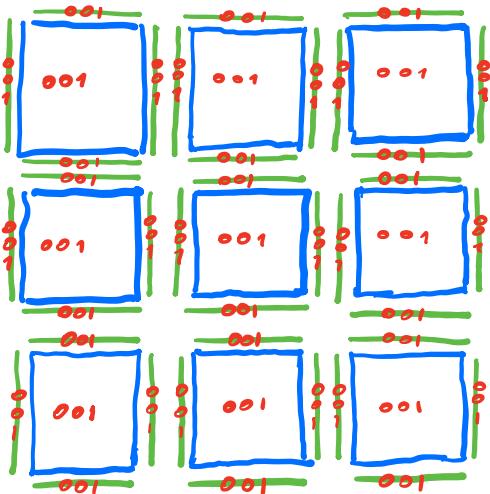
Refined Bit Level 100 and Mask 111



Selection Bit Level 001 and Mask 111



Bit reference levels



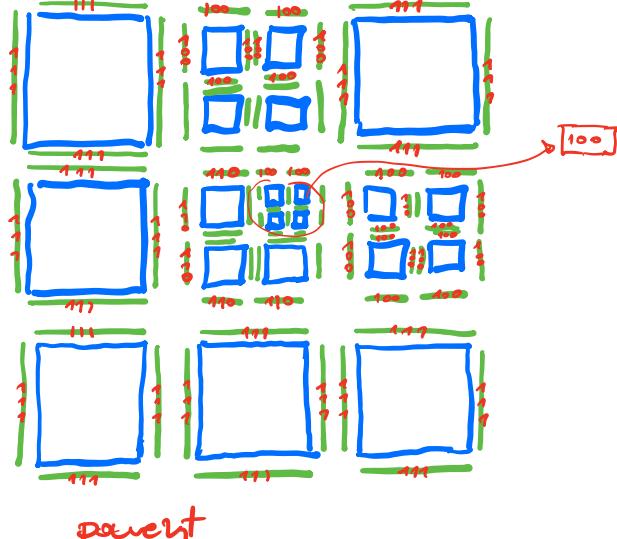
bit = 001
010
100
110
011

mask = 001
010
100

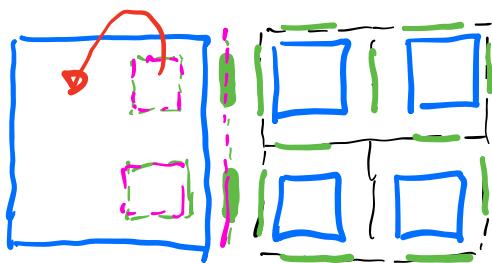
bit & mask

Integration over skeleton

$$\sum_{S \in S} \text{[Eq]} \times \text{vnd}_{\text{up}}^L \partial S$$

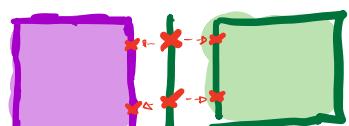
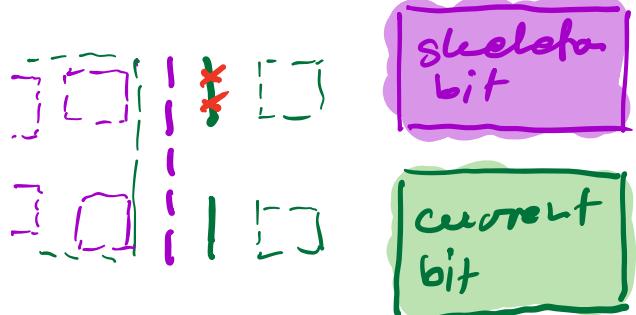


parent

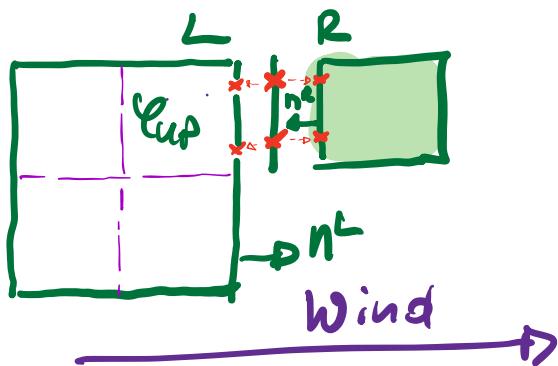


① Skeleton

② Take adj entities
from current
skeleton faces



③ Take parent of skeleton bit adjacent entities.



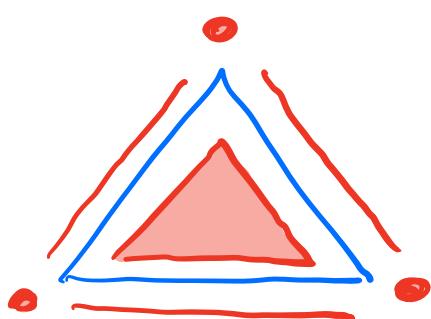
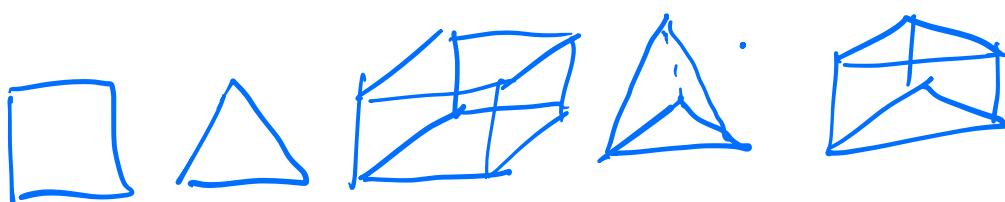
$$\int_S [\delta \varphi] \neq v \cdot n \cdot \varphi_{\text{up}} \text{ d}s$$

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

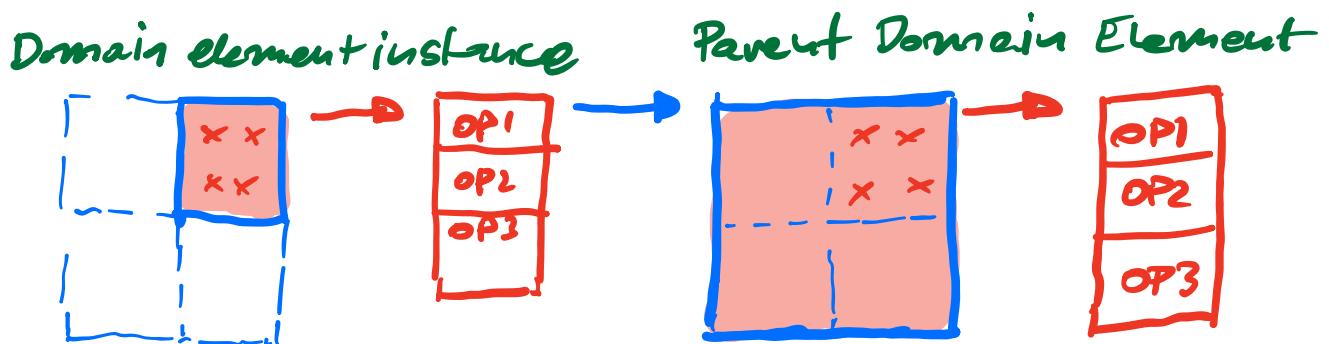
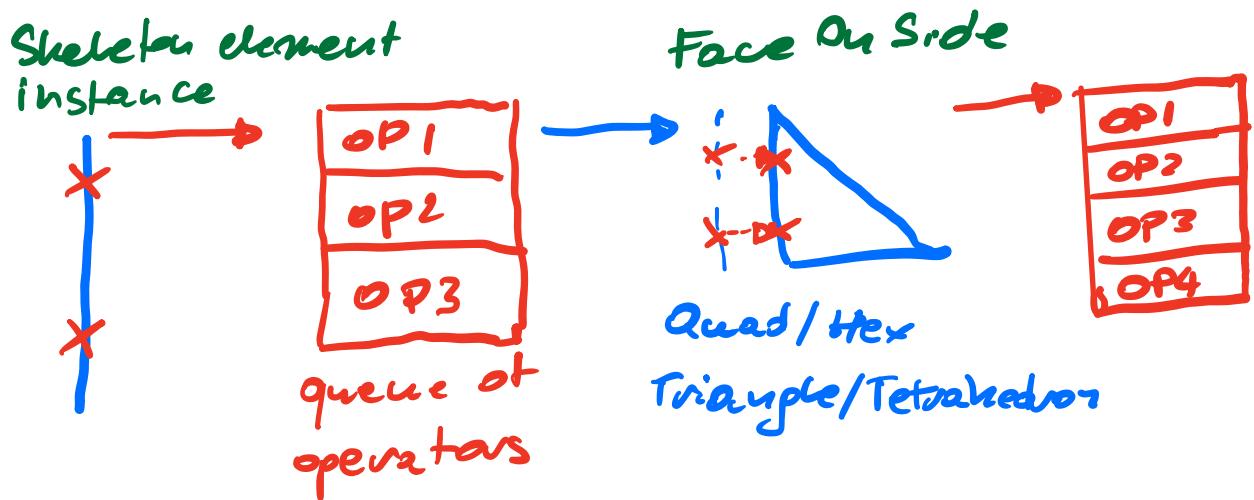
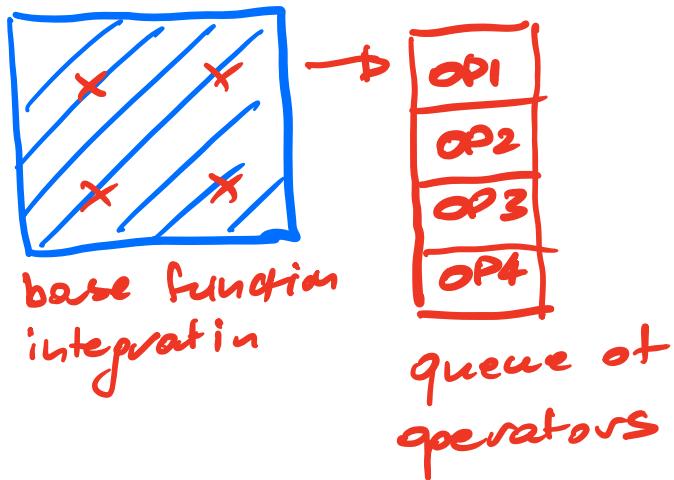
$$\langle v \cdot n \rangle = \frac{1}{2} (v^L \cdot n^L - v^R \cdot n^R)$$

User Data Operators & Finite element instances.

Finite element can be any shape, any type, skeletal 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



```

1 /**
2 * @file level_set.cpp
3 * @example level_set.cpp
4 * @brief Implementation DG upwind method for advection/level set problem
5 * @date 2022-12-15
6 *
7 * @copyright Copyright (c) 2022
8 *
9 */
10
11 #include <MoFEM.hpp>
12 using namespace MoFEM;
13
14 static char help[] = "...\n\n";
15
16 //! [Define dimension]
17 constexpr int SPACE_DIM = EXECUTABLE_DIMENSION;
18 constexpr AssemblyType A = AssemblyType::PETSC; //< selected assembly type
19 constexpr IntegrationType I =
20     IntegrationType::GAUSS; //< selected integration type
21
22 using EntData = EntitiesFieldData::EntData;
23 using DomainEle = PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::DomainEle;
24 using DomianParentEle =
25     PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::DomianParentEle;
26 using BoundaryEle =
27     PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::BoundaryEle;
28 using FaceSideEle =
29     PipelineManager::ElementsAndOpsByDim<SPACE_DIM>::FaceSideEle;
30
31 using DomainEleOp = DomainEle::UserDataOperator;
32 using BoundaryEleOp = BoundaryEle::UserDataOperator;
33 using FaceSideEleOp = FaceSideEle::UserDataOperator;
34
35 using PostProcEle = PostProcBrokenMeshInMoab<DomainEle>;
36
37 constexpr FieldSpace potential_velocity_space = SPACE_DIM == 2 ? H1 : HCURL;
38 constexpr size_t potential_velocity_field_dim = SPACE_DIM == 2 ? 1 : 3;
39
40 constexpr bool debug = false;
41 constexpr int nb_levels = 3; number of refinement levels.
42
43 constexpr int start_bit = nb_levels + 1;
44
45 constexpr int current_bit = 2 * start_bit + 1;
46 constexpr int skeleton_bit = 2 * start_bit + 2;
47 constexpr int aggregate_bit = 2 * start_bit + 3;
48 constexpr int projection_bit = 2 * start_bit + 4;
49 constexpr int aggregate_projection_bit = 2 * start_bit + 5;
50
51 struct LevelSet {
52
53     LevelSet(MoFEM::Interface &m_field) : mField(m_field) {}
54
55     MoFEMErrorCode runProblem();
56
57 private:
58     using VecSideArray = std::array<VectorDouble, 2>;
59     using MatSideArray = std::array<MatrixDouble, 2>;
60
61     /**
62      * @brief data structure carrying information on skeleton on both sides.
63      *
64      */
65     struct SideData {
66         // data for skeleton computation
67         std::array<EntityHandle, 2> feSideHandle;
68         std::array<VectorInt, 2>
69             indicesRowSideMap; //;< indices on rows for left hand-side
70         std::array<VectorInt, 2>
71             indicesColSideMap; //;< indices on columns for left hand-side
72         std::array<MatrixDouble, 2> rowBaseSideMap; // base functions on rows
73         std::array<MatrixDouble, 2> colBaseSideMap; // base function on columns
74         std::array<int, 2> senseMap; // orientation of local element edge/face in
75             // respect to global orientation of edge/face
76
77         VecSideArray lVec; //;< Values of level set field
78         MatSideArray velMat; //;< Values of velocity field
79
80         int currentFEside; //;< current side counter
81     };
82
83     /**
84      * @brief advection velocity field

```

```

87 * normal component.
88 *
89 * \note function define a vector velocity potential field, curl of potential
90 * field gives velocity, thus velocity is divergence free.
91 *
92 * @param SPACE_DIM
93 * @param x
94 * @param y
95 * @param z
96 * @return auto
97 */
98 template <int SPACE_DIM>
99 static double get_velocity_potential(double x, double y, double z);
100 /**
101 * @brief initial level set, i.e. advected filed
102 *
103 * @param x
104 * @param y
105 * @param z
106 * @return double
107 */
108 static double get_level_set(const double x, const double y, const double z);
109 /**
110 * @brief read mesh
111 *
112 * @return MoFEMErrorCode
113 */
114 MoFEMErrorCode readMesh();
115 /**
116 * @brief create fields, and set approximation order
117 *
118 * @return MoFEMErrorCode
119 */
120 MoFEMErrorCode setupProblem();
121 /**
122 * @brief push operators to integrate operators on domain
123 *
124 * @return MoFEMErrorCode
125 */
126 MoFEMErrorCode pushOpDomain();
127 /**
128 * @brief evaluate error
129 *
130 * @return MoFEMErrorCode
131 */
132 std::tuple<double, Tag> evaluateError();
133 /**
134 * @brief Get operator calculating velocity on coarse mesh
135 *
136 * @param vel_ptr
137 * @return DomainEleOp*
138 */
139 ForcesAndSourcesCore::UserDataOperator *
140 getZeroLevelVelOp(boost::shared_ptr<MatrixDouble> vel_ptr);
141 /**
142 * @brief create side element to assemble data from sides
143 *
144 * @param side_data_ptr
145 * @return boost::shared_ptr<FaceSideEle>
146 */
147 boost::shared_ptr<FaceSideEle>
148 getSideFE(boost::shared_ptr<SideData> side_data_ptr);
149 /**
150 * @brief push operator to integrate on skeleton
151 *
152 * @return MoFEMErrorCode
153 */
154 MoFEMErrorCode pushOpSkeleton();
155 /**
156 * @brief test integration side elements
157 *
158 * Check consistency between volume and skeleton integral.
159 *
160 * @return MoFEMErrorCode
161 */
162 MoFEMErrorCode checkConsistency();
163 /**
164 * @brief
165 */
166 /**
167 * @brief
168 */
169 /**
170 */

```

```

173 /**
174 * @brief test consistency between tangent matrix and the right hand side
175 * vectors
176 *
177 * @return MoFEMErrorCode
178 */
179 MoFEMErrorCode testOp();
180
181 /**
182 * @brief initialise field set
183 *
184 * @param level_fun
185 * @return MoFEMErrorCode
186 */
187 MoFEMErrorCode initialiseFieldLevelSet(
188     boost::function<double(double, double, double)> level_fun =
189     get_level_set());
190
191 /**
192 * @brief initialise potential velocity field
193 *
194 * @param vel_fun
195 * @return MoFEMErrorCode
196 */
197 MoFEMErrorCode initialiseFieldVelocity(
198     boost::function<double(double, double, double)> vel_fun =
199     get_velocity_potential<SPACE_DIM>());
200
201 /**
202 * @brief dg level set projection
203 *
204 * @param prj_bit
205 * @param mesh_bit
206 * @return MoFEMErrorCode
207 */
208 MoFEMErrorCode dgProjection(const int prj_bit = projection_bit);
209
210 /**
211 * @brief solve advection problem
212 *
213 * @return * MoFEMErrorCode
214 */
215 MoFEMErrorCode solveAdvection();
216
217 /**
218 * @brief Wrapper executing stages while mesh refinement
219 */
220 struct WrapperClass {
221     WrapperClass() = default;
222     virtual MoFEMErrorCode setBits(LevelSet &level_set, int l) = 0;
223     virtual MoFEMErrorCode runCalcs(LevelSet &level_set, int l) = 0;
224     virtual MoFEMErrorCode setAggregateBit(LevelSet &level_set, int l) = 0;
225     virtual double getThreshold(const double max) = 0;
226 };
227
228 /**
229 * @brief Used to execute initial mesh approximation while mesh refinement
230 *
231 */
232 struct WrapperClassInitialSolution : public WrapperClass {
233
234     WrapperClassInitialSolution(boost::shared_ptr<double> max_ptr)
235         : WrapperClass(), maxPtr(max_ptr) {}
236
237     MoFEMErrorCode setBits(LevelSet &level_set, int l) {
238         MoFEMFunctionBegin;
239         auto simple = level_set.mField.getInterface<Simple>();
240         simple->getBitRefLevel() =
241             BitRefLevel().set(skeleton_bit) | BitRefLevel().set(aggregate_bit);
242         simple->getBitRefLevelMask() = BitRefLevel().set();
243         simple->reSetUp(true);
244         MoFEMFunctionReturn(0);
245     };
246
247     MoFEMErrorCode runCalcs(LevelSet &level_set, int l) {
248         MoFEMFunctionBegin;
249         CHKERR level_set.initialiseFieldLevelSet();
250         MoFEMFunctionReturn(0);
251     }
252
253     MoFEMErrorCode setAggregateBit(LevelSet &level_set, int l) {
254         auto bit_mng = level_set.mField.getInterface<BitRefManager>();
255         auto set_bit = [] (auto l) { return BitRefLevel().set(l); };
256         MoFEMFunctionBegin;

```

```

259             BitRefLevel().set(), level);
260     CHKERR level_set.mField.getInterface<CommInterface>()
261         ->synchroniseEntities(level);
262     CHKERR bit_mng->setNthBitRefLevel(current_bit, false);
263     CHKERR bit_mng->setNthBitRefLevel(level, current_bit, true);
264     CHKERR bit_mng->setNthBitRefLevel(level, aggregate_bit, true);
265     MoFEMFunctionReturn(0);
266 }
267
268 double getThreshold(const double max) {
269     *maxPtr = std::max(*maxPtr, max);
270     return 0.05 * (*maxPtr);
271 }
272
273 private:
274     boost::shared_ptr<double> maxPtr;
275 };
276
277 /**
278  * @brief Use peculated errors on all levels while mesh projection
279  *
280  */
281 struct WrapperClassErrorProjection : public WrapperClass {
282     WrapperClassErrorProjection(boost::shared_ptr<double> max_ptr)
283         : maxPtr(max_ptr) {}
284
285     MoFEMErrorCode setBits(LevelSet &level_set, int l) { return 0; }
286     MoFEMErrorCode runCalcs(LevelSet &level_set, int l) { return 0; }
287     MoFEMErrorCode setAggregateBit(LevelSet &level_set, int l) {
288         auto bit_mng = level_set.mField.getInterface<BitRefManager>();
289         auto set_bit = [] (auto l) { return BitRefLevel().set(l); };
290         MoFEMFunctionBegin;
291         Range level;
292         CHKERR bit_mng->getEntitiesByRefLevel(set_bit(start_bit + l),
293                                              BitRefLevel().set(), level);
294         CHKERR level_set.mField.getInterface<CommInterface>()
295             ->synchroniseEntities(level);
296         CHKERR bit_mng->setNthBitRefLevel(current_bit, false);
297         CHKERR bit_mng->setNthBitRefLevel(level, current_bit, true);
298         CHKERR bit_mng->setNthBitRefLevel(level, aggregate_bit, true);
299         MoFEMFunctionReturn(0);
300     }
301     double getThreshold(const double max) { return 0.05 * (*maxPtr); }
302
303 private:
304     boost::shared_ptr<double> maxPtr;
305 };
306
307 MoFEMErrorCode refineMesh(WrapperClass &&wp);
308
309 struct OpRhsDomain; //< integrate volume operators on rhs
310 struct OpLhsDomain; //< integrate volume operator on lhs
311 struct OpRhsSkeleton; //< integrate skeleton operators on rhs
312 struct OpLhsSkeleton; //< integrate skeleton operators on khs
313
314 // Main interfaces
315 MoFEM::Interface &mField;
316
317 using AssemblyDomainEleOp =
318     FormsIntegrators<DomainEleOp>::Assembly<A>::OpBase;
319 using OpMassLL =
320     FormsIntegrators<DomainEleOp>::Assembly<A>::BiLinearForm<I>::OpMass<1, 1>;
321 using OpSourceL =
322     FormsIntegrators<DomainEleOp>::Assembly<A>::LinearForm<I>::OpSource<1, 1>;
323 using OpMassVV = FormsIntegrators<DomainEleOp>::Assembly<A>::BiLinearForm<
324     I>::OpMass<potential_velocity_field_dim, potential_velocity_field_dim>;
325 using OpSourceV = FormsIntegrators<DomainEleOp>::Assembly<A>::LinearForm<
326     I>::OpSource<potential_velocity_field_dim, potential_velocity_field_dim>;
327 using OpScalarFieldL = FormsIntegrators<DomainEleOp>::Assembly<A>::LinearForm<
328     I>::OpBaseTimesScalar<1>;
329
330 using AssemblyBoundaryEleOp =
331     FormsIntegrators<BoundaryEleOp>::Assembly<A>::OpBase;
332
333 enum ElementSide { LEFT_SIDE = 0, RIGHT_SIDE = 1 };
334
335 private:
336     boost::shared_ptr<double> maxPtr;
337 };
338
339 };
340
341 template <>
342 double LevelSet::get_velocity_potential<2>(double x, double y, double z) {

```

```

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(WrapperClassInitialSolution(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(); Run advection problem
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     ... ...
430

```

Check consistency of tangent matrix and right hand side.

Initial mesh refinement.

```

431
432 int main(int argc, char *argv[]) {
433
434     // Initialisation of MoFEM/PETSc and MOAB data structures
435     const char param_file[] = "param_file.petsc";
436     MoFEM::Core::Initialize(&argc, &argv, param_file, help);
437
438     try {
439
440         // Create MoAB database
441         moab::Core moab_core;
442         moab::Interface &moab = moab_core;
443
444         // Create MoFEM database and link it to MoAB
445         MoFEM::Core mofem_core(moab);
446         MoFEM::Interface &m_field = mofem_core;
447
448         // Register DM Manager
449         DMTType dm_name = "DMMOFEM";
450         CHKERRQ DMRegister_MoFEM(dm_name);
451
452         // Add logging channel for example
453         auto core_log = logging::core::get();
454         core_log->add_sink(
455             LogManager::createSink(LogManager::getStrmWorld(), "LevelSet"));
456         LogManager::setLog("LevelSet");
457         MOFEM_LOG_TAG("LevelSet", "LevelSet");
458
459         LevelSet level_set(m_field);
460         CHKERRQ level_set.runProblem();
461     }
462     CATCH_ERRORS;
463
464     // finish work cleaning memory, getting statistics, etc.
465     MoFEM::Core::Finalize();
466
467     return 0;
468 }
```

```

469 MoFEMErrorCode LevelSet::readMesh() {
470     MoFEMFunctionBegin;
471     auto simple = mField.getInterface<Simple>();
472     // get options from command line
473     CHKERR simple->getOptions();
474
475     // Only L2 field is set in this example. Two lines bellow forces simple
476     // interface to creat lower dimension (edge) elements, despite that fact that
477     // there is no field spanning on such elements. We need them for DG method.
478     simple->getAddSkeletonFE() = true;
479     simple->getAddBoundaryFE() = true;
480
481     // load mesh file
482     simple->getBitRefLevel() = BitRefLevel();
483     CHKERR simple->loadFile();
484
485     auto set_problem_bit = [&]() {
486         MoFEMFunctionBegin;
487         auto bit0 = BitRefLevel().set(start_bit);
488         BitRefLevel start_mask;
489         for (auto s = 0; s != start_bit; ++s)
490             start_mask[s] = true;
491
492         auto bit_mng = mField.getInterface<BitRefManager>();
493
494         Range level0;
495         CHKERR bit_mng->getEntitiesByRefLevel(BitRefLevel().set(0),
496                                               BitRefLevel().set(), level0);
497                                               , true);
498         CHKERR bit_mng->setNthBitRefLevel(level0, current_bit, true);
499         CHKERR bit_mng->setNthBitRefLevel(level0, aggregate_bit, true);
500         CHKERR bit_mng->setNthBitRefLevel(level0, skeleton_bit, true);
501
502         simple->getBitAdjEnt() = BitRefLevel().set();
503         simple->getBitAdjParent() = BitRefLevel().set();
504         simple->getBitRefLevel() = BitRefLevel().set(current_bit);
505         simple->getBitRefLevelMask() = BitRefLevel().set();
506
507 #ifndef NDEBUG
508     if constexpr (debug) {
509         auto proc_str = boost::lexical_cast<std::string>(mField.get_comm_rank());
510         CHKERR bit_mng->writeBitLevelByDim(
511             BitRefLevel().set(0), BitRefLevel().set(), SPACE_DIM,
512             (proc_str + "level_base.vtk").c_str(), "VTK", "");
513     }
514 
```

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 adjacent to each other
`t_comm_rank());`

```

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;

```

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

Integrate "rhs" on domain elements.

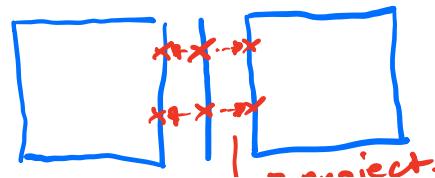
$$t = \sum_i w_i (\sigma + \alpha \cdot r_i) - \frac{\partial \delta \varphi}{\partial x_i} \cdot v(\zeta_i) \cdot \varphi(\zeta_i)$$

calculate tangent matrix at domain element.

Calculate "rhs" on
skeleton.

Iterate side elements & their parents to get data on base functions, indices, and field values.

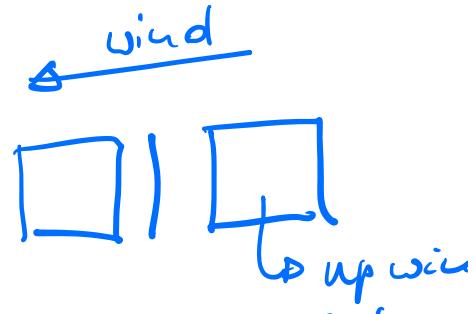
Integration points are projected as elements on the side of skeleton



```

689 // get 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 MoFEMFunctionReturn(0);
760 }
761
762 MoFEMErrorCode
763 LevelSet::OpLhsSkeleton::doWork(int side, EntityType type,
764                                 EntitiesFieldData::EntData &data) {
765     MoFEMFunctionBegin;
766
767     // Collect data from side domain elements
768     CHKERR loopSideFaces("dFE", *sideFEPtr);
769     const auto in_the_loop =
770         sideFEPtr->nInTheLoop; // return number of elements on the side
771
772     . . .

```



↑ select upwind side

$$\sum_i w_i [\delta \varphi(z_i)] \left(\nabla z_i \cdot n \right) \varphi_u$$

→ assemble "rhs"

calculate tangent matrix on skeleton.

```

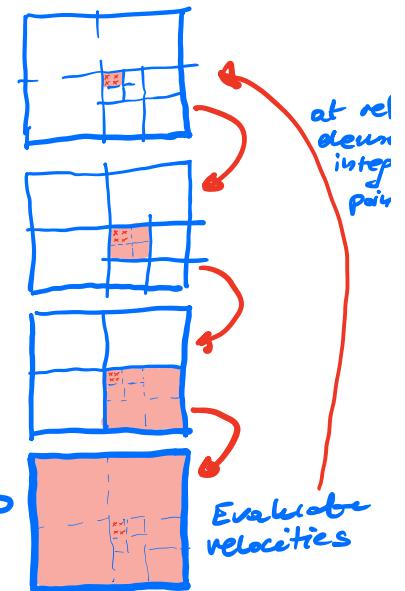
775 };
776
777 auto get_ntensor = [] (auto &base_mat) {
778     return FTensor::Tensor0<FTensor::PackPtr<double *, 1>>(
779         &*base_mat.data().begin());
780 };
781
782 if (in_the_loop > 0) {
783
784     // get normal of the face or edge
785     auto t_normal = getFTensor1Normal();
786     const 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                     ...
860                 );
861             }
862         }
863     }
864 }
```

```

861             &*sideDataPtr->indicesColSideMap[s1].begin(),
862             &*matSkeleton.data().begin(), ADD_VALUES);
863     }
864   }
865 }
866 MoFEMFunctionReturn(0);
867 }
868
869 ForcesAndSourcesCore::UserDataOperator *
870 LevelSet::getZeroLevelVelOp(boost::shared_ptr<MatrixDouble> vel_ptr) {
871   auto get_parent_vel_this = [&]() {
872     auto parent_fe_ptr = boost::make_shared<DomianParentEle>(mField);
873     CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
874       parent_fe_ptr->getOpPtrVector(), {potential_velocity_space});
875     parent_fe_ptr->getOpPtrVector().push_back(
876       new OpCalculateHcurlVectorCurl<potential_velocity_field_dim, SPACE_DIM>(
877         "V", vel_ptr));
878   return parent_fe_ptr;
879 }; calculate curl of velocity potential field
880
881 auto get_parents_vel_fe_ptr = [&](auto this_fe_ptr) {
882   std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
883   for (int l = 0; l <= nb_levels; ++l)
884     parents_elems_ptr_vec.emplace_back(
885       boost::make_shared<DomianParentEle>(mField));
886   for (auto l = 1; l <= nb_levels; ++l) {
887     parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
888       new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
889                      BitRefLevel().set(0).flip(), this_fe_ptr,
890                      BitRefLevel().set(0), BitRefLevel().set()));
891   }
892   return parents_elems_ptr_vec[0];
893 };
894
895 auto this_fe_ptr = get_parent_vel_this();
896 auto parent_fe_ptr = get_parents_vel_fe_ptr(this_fe_ptr);
897 return new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
898                        BitRefLevel().set(0).flip(), this_fe_ptr,
899                        BitRefLevel().set(0), BitRefLevel().set());
900 }
901

```

Get operator calculating velocities on parent element.



```

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 AddH0Ops<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 AddH0Ops<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 }

```

push operators to integrate over domain elements.

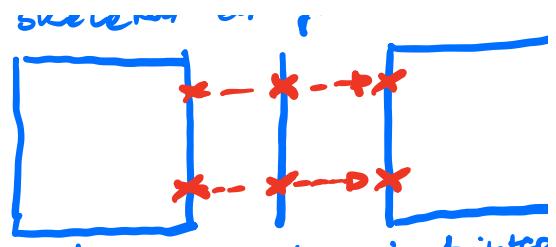
Integrate only on element with current bit reference level on.

... side element

```

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

```



Side element project integration points from skeleton to side.

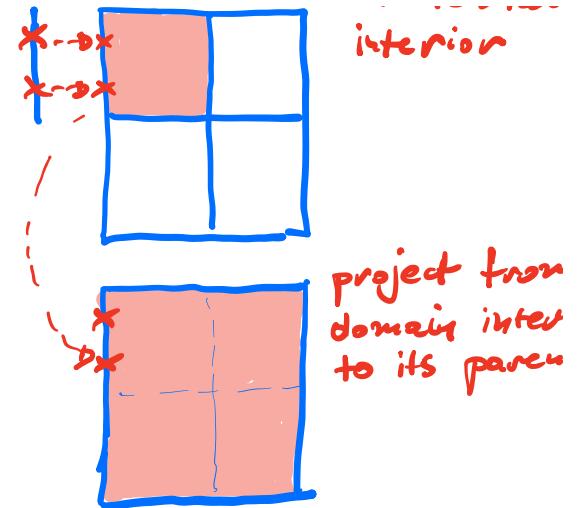
Clear structure storing data from element on skeleton side.

Collect data on skeleton side.

```

1033         "Wrong number of integration 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 AddH0Ops<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;

```



Calculate velocity and level set values at integration points.

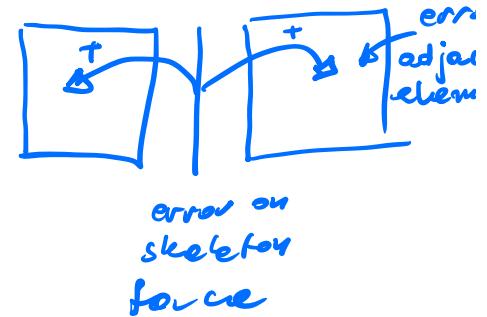
Traverse parent until element with current bit is set.

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->numericalEntFiniteElementPtr->getBitRefLevel().test(
1134             skeleton_bit);
1135     };
1136     pip->getSkeletonRhsFE()->exeTestHook = [&](FEMethod *fe_ptr) {
1137         return fe_ptr->numericalEntFiniteElementPtr->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         }
1204     };
1205 }
```

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



↳ calculate jump on skeleton

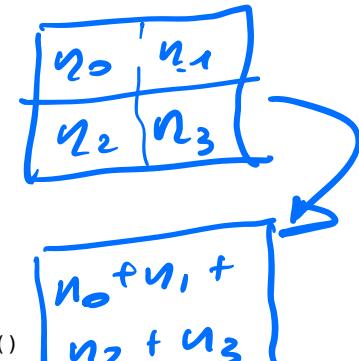
```

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->numericalFiniteElementPtr->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->numericalFiniteElementPtr->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->numericalFiniteElementPtr->getEnt();
1287         auto parent = fe_ptr->numericalFiniteElementPtr->getParentEnt();
1288         auto th_parent = fe_ptr->numericalFiniteElementPtr->getBasicDataPtr()
1289

```

side element

← project error from
children to parents.

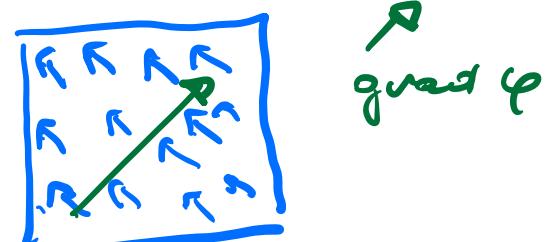


```

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
1349     MoFEMErrorCode doWork(int side, EntityType type,
1350                           EntitiesFieldData::EntData &data) {
1351         MoFEMFunctionBegin;
1352         const auto nb_dofs = data.getIndices().size();
1353         if (nb_dofs) {
1354             const auto nb_gauss_pts = getGaussPts().size2();
1355             const auto t_w = getFTensor0IntegrationWeight();
1356             auto t_diff = data.getFTensor1DiffN<SPACE_DIM>();
1357             auto t_l = getFTensor0FromVec(*lPtr);
1358             auto t_vel = getFTensor1FromMat<SPACE_DIM>(*velPtr);
1359             double div = 0;
1360             for (auto gg = 0; gg != nb_gauss_pts; ++gg) {
1361                 for (int rr = 0; rr != nb_dofs; ++rr) {
1362                     div += getMeasure() * t_w * t_l * (t_diff(i) * t_vel(i));
1363                     ++t_diff;
1364                 }
1365                 ++t_w;
1366                 ++t_l;
1367                 ++t_vel;
1368             }
1369             CHKERR VecSetValue(divVec, 0, div, ADD_VALUES);
1370         }
1371         MoFEMFunctionReturn(0);
1372     }
1373
1374     private:
1375         boost::shared_ptr<VectorDouble> lPtr;

```

Test consistency between
volume and skeleton
integral



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

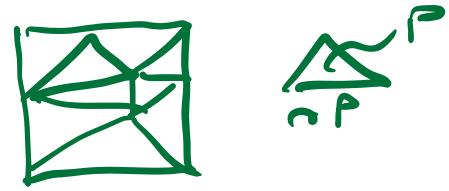
```

1377 };
1378
1379 /**
1380 * @brief calculate skeleton integral
1381 *
1382 */
1383 struct DivergenceSkeleton : public BoundaryEleOp {
1384     DivergenceSkeleton(boost::shared_ptr<SideData> side_data_ptr,
1385                         boost::shared_ptr<FaceSideEle> side_fe_ptr,
1386                         SmartPetscObj<Vec> div_vec)
1387         : BoundaryEleOp(NOSPACE, BoundaryEleOp::OPSPACE),
1388           sideDataPtr(side_data_ptr), sideFEPtr(side_fe_ptr), divVec(div_vec) {}
1389     MoFEMErrorCode doWork(int side, EntityType type,
1390                           EntitiesFieldData::EntData &data) {
1391         MoFEMFunctionBegin;
1392
1393         auto get_ntensor = [] (auto &base_mat) {
1394             return FTensor::Tensor0<FTensor::PackPtr<double *, 1>>(
1395                 &*base_mat.data().begin());
1396         };
1397
1398         auto not_side = [] (auto s) {
1399             return s == LEFT_SIDE ? RIGHT_SIDE : LEFT_SIDE;
1400         };
1401
1402         // Collect data from side domain elements
1403         CHKERR loopSideFaces("dFE", *sideFEPtr);
1404         const auto in_the_loop =
1405             sideFEPtr->nInTheLoop; // return number of elements on the side
1406
1407         auto t_normal = getFTensor1Normal();
1408         const auto nb_gauss_pts = getGaussPts().size2();
1409         for (auto s0 : {LEFT_SIDE, RIGHT_SIDE}) {
1410             const auto nb_dofs = sideDataPtr->indicesRowSideMap[s0].size();
1411             if (nb_dofs) {
1412                 auto t_base = get_ntensor(sideDataPtr->rowBaseSideMap[s0]);
1413                 auto nb_row_base_functions = sideDataPtr->rowBaseSideMap[s0].size2();
1414                 auto side_sense = sideDataPtr->senseMap[s0];
1415                 auto opposite_s0 = not_side(s0);
1416
1417                 auto arr_t_l =
1418                     make_array(getFTensor0FromVec(sideDataPtr->lVec[LEFT_SIDE]),
1419                                getFTensor0FromVec(sideDataPtr->lVec[RIGHT_SIDE]));
1420                 auto arr_t_vel = make_array(
1421                     getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[LEFT_SIDE]),
1422                     getFTensor1FromMat<SPACE_DIM>(sideDataPtr->velMat[RIGHT_SIDE]));
1423
1424                 auto next = [&]() {
1425                     for (auto &t_l : arr_t_l)
1426                         ++t_l;
1427                     for (auto &t_vel : arr_t_vel)
1428                         ++t_vel;
1429                 };
1430
1431                 double div = 0;
1432
1433                 auto t_w = getFTensor0IntegrationWeight();
1434                 for (int gg = 0; gg != nb_gauss_pts; ++gg) {
1435                     FTensor::Tensor1<double, SPACE_DIM> t_vel;
1436                     t_vel(i) =
1437                         (arr_t_vel[LEFT_SIDE](i) + arr_t_vel[RIGHT_SIDE](i)) / 2.;
1438                     const auto dot = (t_normal(i) * t_vel(i)) * side_sense;
1439                     const auto l_upwind_side = (dot > 0) ? s0 : opposite_s0;
1440                     const auto l_upwind =
1441                         arr_t_l[l_upwind_side]; //< assume that field is continues,
1442                                         // initialisation field has to be smooth
1443                                         // and exactly approximated by approx
1444                                         // base
1445                     auto res = t_w * l_upwind * dot;
1446                     ++t_w;
1447                     next();
1448                     int rr = 0;
1449                     for (; rr != nb_dofs; ++rr) {
1450                         div += t_base * res;
1451                         ++t_base;
1452                     }
1453                     for (; rr < nb_row_base_functions; ++rr) {
1454                         ++t_base;
1455                     }
1456                 }
1457                 CHKERR VecSetValue(divVec, 0, div, ADD_VALUES);
1458             }
1459             if (!in_the_loop)
1460                 break;
1461         }

```

$$\int_{\Omega} \delta \varphi u \cdot v_i \nabla u \cdot \nabla v_i d\Omega = \int_{\partial\Omega} \delta \varphi u \cdot v_i \nabla u \cdot \nabla v_i \cdot \nu d\Gamma$$

if $\nu \cdot \nabla \varphi \perp \nu$



```

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 AddH0Ops<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

```

Generate fields

Calculate integrals for domain, skeleton and boundary

```

1549 MoFEMErrorCode LevelSet::testOp() {
1550     MoFEMFunctionBegin;
1551
1552     // get operators tester
1553     auto simple = mField.getInterface<Simple>();
1554     auto opt = mField.getInterface<OperatorsTester>(); // get interface to
1555                                         // OperatorsTester
1556     auto pip = mField.getInterface<PipelineManager>(); // get interface to
1557                                         // pipeline manager
1558
1559     CHKERR pushOpDomain();                                push operators
1560     CHKERR pushOpSkeleton();                            used to print inconsisti
1561
1562     auto post_proc = [&](auto dm, auto f_res, auto out_name) {
1563         MoFEMFunctionBegin;
1564         auto post_proc_fe =
1565             boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
1566
1567         using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
1568
1569         auto l_vec = boost::make_shared<VectorDouble>();
1570         post_proc_fe->getOpPtrVector().push_back(
1571             new OpCalculateScalarFieldValues("L", l_vec, f_res));
1572
1573         post_proc_fe->getOpPtrVector().push_back(
1574
1575             new OpPPMap(
1576
1577                 post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),
1578
1579                 {{"L", l_vec}},
1580
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());

```

test Tangent matrix
elements, skeleton and domain

Generate random fields

Validate directid
directive.

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

```

1635     MoFEMFunctionReturn(0);
1636 }
1637 }
1638 MoFEMErrorCode LevelSet::initialiseFieldLevelSet(
1639     boost::function<double(double, double, double)> level_fun) {
1640     MoFEMFunctionBegin;
1641
1642     // get operators tester
1643     auto simple = mField.getInterface<Simple>();
1644     auto pip = mField.getInterface<PipelineManager>(); // get interface to
1645                                         // pipeline manager
1646     auto prb_mng = mField.getInterface<ProblemsManager>();
1647
1648     boost::shared_ptr<FEMethod> lhs_fe = boost::make_shared<DomainEle>(mField);
1649     boost::shared_ptr<FEMethod> rhs_fe = boost::make_shared<DomainEle>(mField);
1650     auto swap_fe = [&]() {
1651         lhs_fe.swap(pip->getDomainLhsFE());
1652         rhs_fe.swap(pip->getDomainRhsFE());
1653     };
1654     swap_fe();
1655
1656     pip->setDomainLhsIntegrationRule([](int, int, int o) { return 3 * o; });
1657     pip->setDomainRhsIntegrationRule([](int, int, int o) { return 3 * o; });
1658
1659     auto sub_dm = createSmartDM(mField.get_comm(), "DMMOFEM");
1660     CHKERR DMMoFEMCreateSubDM(sub_dm, simple->getDM(), "LEVELSET_POJECTION");
1661     CHKERR DMMoFEMSetDestroyProblem(sub_dm, PETSC_TRUE);
1662     CHKERR DMMoFEMSetSquareProblem(sub_dm, PETSC_TRUE);
1663     CHKERR DMMoFEMAddElement(sub_dm, simple->getDomainFEName());
1664     CHKERR DMMoFEMAddSubFieldRow(sub_dm, "L");
1665     CHKERR DMSetUp(sub_dm);
1666
1667     BitRefLevel remove_mask = BitRefLevel().set(current_bit);
1668     remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
1669     CHKERR prb_mng->removeDofsOnEntities("LEVELSET_POJECTION", "L",
1670                                         BitRefLevel().set(), remove_mask);
1671     auto test_bit_ele = [&](FEMethod *fe_ptr) {
1672         return fe_ptr->numericalFiniteElementPtr->getBitRefLevel().test(
1673             current_bit);
1674     };
1675     pip->getDomainLhsFE()->exeTestHook = test_bit_ele;
1676     pip->getDomainRhsFE()->exeTestHook = test_bit_ele;
1677
1678     CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1679         pip->getOpDomainRhsPipeline(), {potential_velocity_space, L2});
1680     CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1681         pip->getOpDomainLhsPipeline(), {potential_velocity_space, L2});
1682     pip->getOpDomainLhsPipeline().push_back(new OpMassLL("L", "L"));
1683     pip->getOpDomainRhsPipeline().push_back(new OpSourceL("L", level_fun));
1684
1685     CHKERR mField.getInterface<FieldBlas>()->setField(0, "L");
1686
1687     auto ksp = pip->createKSP(sub_dm);
1688     CHKERR KSPSetDM(ksp, sub_dm);
1689     CHKERR KSPSetFromOptions(ksp);
1690     CHKERR KSPSetUp(ksp);
1691
1692     auto L = smartCreateDMVector(sub_dm);
1693     auto F = smartVectorDuplicate(L);
1694
1695     CHKERR KSPSolve(ksp, F, L);
1696     CHKERR VecGhostUpdateBegin(L, INSERT_VALUES, SCATTER_FORWARD);
1697     CHKERR VecGhostUpdateEnd(L, INSERT_VALUES, SCATTER_FORWARD);
1698     CHKERR DMoFEMMeshToLocalVector(sub_dm, L, INSERT_VALUES, SCATTER_REVERSE);
1699
1700     auto [error, th_error] = evaluateError();
1701     MOFEM_LOG("LevelSet", Sev::inform) << "Error indicator " << error;
1702 #ifndef NDEBUG
1703     auto fe_meshset =
1704         mField.get_finite_element_meshset(simple->getDomainFEName());
1705     std::vector<Tag> tags{th_error};
1706     CHKERR mField.get_moab().write_file("error.h5m", "MOAB",
1707                                         "PARALLEL=WRITE_PART", &fe_meshset, 1,
1708                                         &tags.begin(), tags.size());
1709 #endif
1710
1711     auto post_proc = [&](auto dm, auto out_name, auto th_error) {
1712         MoFEMFunctionBegin;
1713         auto post_proc_fe =
1714             boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
1715         post_proc_fe->setTagsToTransfer({th_error});
1716         post_proc_fe->exeTestHook = test_bit_ele;
1717
1718         . . .
1719     };

```

Initialise level set field

```

1721 auto l_vec = boost::make_shared<VectorDouble>();
1722 auto l_grad_mat = boost::make_shared<MatrixDouble>();
1723 CHKERR AddH0Ops<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 MoFEMErrorCode LevelSet::initialiseFieldVelocity(
1758     boost::function<double(double, double, double)> vel_fun) {
1759     MoFEMFunctionBegin;
1760
1761 // get operators tester
1762 auto simple = mField.getInterface<Simple>();
1763 auto pip = mField.getInterface<PipelineManager>(); // get interface to
1764 // pipeline manager
1765 auto prb_mng = mField.getInterface<ProblemsManager>();
1766
1767 boost::shared_ptr<FEMethod> lhs_fe = boost::make_shared<DomainEle>(mField);
1768 boost::shared_ptr<FEMethod> rhs_fe = boost::make_shared<DomainEle>(mField);
1769 auto swap_fe = [&]() {
1770     lhs_fe.swap(pip->getDomainLhsFE());
1771     rhs_fe.swap(pip->getDomainRhsFE());
1772 };
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 coarse 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->numericalFiniteElementPtr->getBitRefLevel().test(0);
1795 };
1796 pip->getDomainLhsFE()->exeTestHook = test_bit;
1797 pip->getDomainRhsFE()->exeTestHook = test_bit;
1798
1799 CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1800     pip->getOpDomainLhsPipeline(), {potential_velocity_space, L2});
1801 CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1802     pip->getOpDomainRhsPipeline(), {potential_velocity_space, L2});
1803
1804 pip->getOpDomainLhsPipeline().push_back(new OpMassVV("V", "V"));

```

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

```

1807 auto ksp = pip->createKSP(sub_dm);
1808 CHKERR KSPSetDM(ksp, sub_dm);
1809 CHKERR KSPSetFromOptions(ksp);
1810 CHKERR KSPSetUp(ksp);
1811
1812 auto L = smartCreateDMVector(sub_dm);
1813 auto F = smartVectorDuplicate(L);
1814
1815 CHKERR KSPSolve(ksp, F, L);
1816 CHKERR VecGhostUpdateBegin(L, INSERT_VALUES, SCATTER_FORWARD);
1817 CHKERR VecGhostUpdateEnd(L, INSERT_VALUES, SCATTER_FORWARD);
1818 CHKERR DMoFEMMeshToLocalVector(sub_dm, L, INSERT_VALUES, SCATTER_REVERSE);
1819
1820 auto post_proc = [&](auto dm, auto out_name) {
1821     MoFEMFunctionBegin;
1822     auto post_proc_fe =
1823         boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
1824     post_proc_fe->exeTestHook = test_bit;
1825
1826     CHKERR AddH00ps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1827         post_proc_fe->getOpPtrVector(), {potential_velocity_space});
1828
1829     using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
1830
1831     if constexpr (SPACE_DIM == 2) {
1832         auto l_vec = boost::make_shared<VectorDouble>();
1833         auto potential_vec = boost::make_shared<VectorDouble>();
1834         auto velocity_mat = boost::make_shared<MatrixDouble>();
1835
1836         post_proc_fe->getOpPtrVector().push_back(
1837             new OpCalculateScalarFieldValues("V", potential_vec));
1838         post_proc_fe->getOpPtrVector().push_back(
1839             new OpCalculateHcurlVectorCurl<potential_velocity_field_dim,
1840                                         SPACE_DIM>("V", velocity_mat));
1841
1842         post_proc_fe->getOpPtrVector().push_back(
1843
1844             new OpPPMap(
1845
1846                 post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),
1847
1848                 {{"VelocityPotential", potential_vec}},
1849
1850                 {{"Velocity", velocity_mat}},
1851
1852                 {}, {}));
1853
1854     );
1855
1856 } else {
1857     SETERRQ(PETSC_COMM_SELF, MOFEM_NOT_IMPLEMENTED,
1858             "3d case not implemented");
1859 }
1860
1861 CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
1862                                     post_proc_fe);
1863 post_proc_fe->writeFile(out_name);
1864 MoFEMFunctionReturn(0);
1865 };
1866
1867 if constexpr (debug)
1868     CHKERR post_proc(sub_dm, "initial_velocity_potential.h5m");
1869
1870 swap_fe();
1871
1872 MoFEMFunctionReturn(0);
1873 }
1874
1875 LevelSet *level_set_raw_ptr = nullptr;
1876 MoFEM::TsCtx *ts_ctx;
1877
1878 MoFEMErrorCode LevelSet::solveAdvection() {
1879     MoFEMFunctionBegin;
1880
1881     // get operators tester
1882     auto simple = mField.getInterface<Simple>();
1883     auto pip = mField.getInterface<PipelineManager>(); // get interface to
1884     auto prb_mng = mField.getInterface<ProblemsManager>();
1885
1886     CHKERR pushOpDomain();
1887     CHKERR pushOpSkeleton();
1888
1889     auto sub_dm = createSmartDM(mField.get_comm(), "DMMOFEM");
1890     CHKERR DMMOFEMCreateSubDM(sub_dm, simple->getDM(), "ADVECTION");
1891     CHKERR DMM_OFEM_ID(mField.get_comm(), sub_dm, PETSC_TRUE);
1892
1893     MoFEMFunctionReturn(0);
1894 }
```

Solve advection problem

```

1893 CHKERR DMMoFEMAddElement(sub_dm, simple->getDomainFEName());
1894 CHKERR DMMoFEMAddElement(sub_dm, simple->getSkeletonFEName());
1895 CHKERR DMMoFEMAddSubFieldRow(sub_dm, "L");
1896 CHKERR DMSetUp(sub_dm);
1897
1898 BitRefLevel remove_mask = BitRefLevel().set(current_bit);
1899 remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
1900 CHKERR prb_mng->removeDofsOnEntities("ADVECTION", "L", BitRefLevel().set(),
1901                                         remove_mask);
1902
1903 auto add_post_proc_fe = [&]() {
1904     auto post_proc_fe = boost::make_shared<PostProcEle>(mField);
1905
1906     Tag th_error;
1907     double def_val = 0;
1908     CHKERR mField.get_moab().tag_get_handle(
1909         "Error", 1, MB_TYPE_DOUBLE, th_error, MB_TAG_CREAT | MB_TAG_SPARSE,
1910         &def_val);
1911     post_proc_fe->setTagsToTransfer({th_error});
1912
1913     post_proc_fe->exeTestHook = [&](FEMethod *fe_ptr) {
1914         return fe_ptr->numericalFiniteElementPtr->getBitRefLevel().test(
1915             current_bit);
1916     };
1917
1918     using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
1919     auto l_vec = boost::make_shared<VectorDouble>();
1920     auto vel_ptr = boost::make_shared<MatrixDouble>();
1921
1922     CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
1923         post_proc_fe->getOpPtrVector(), {H1});
1924     post_proc_fe->getOpPtrVector().push_back(
1925         new OpCalculateScalarFieldValues("L", l_vec));
1926     post_proc_fe->getOpPtrVector().push_back(getZeroLevelVelOp(vel_ptr));
1927
1928     post_proc_fe->getOpPtrVector().push_back(
1929
1930         new OpPPMap(
1931             post_proc_fe->getPostProcMesh(),
1932
1933             post_proc_fe->getMapGaussPts(),
1934
1935             {{"L", l_vec}},
1936
1937             {{"V", vel_ptr}},
1938
1939             {}, {}
1940
1941     );
1942     return post_proc_fe;
1943 };
1944
1945
1946 auto post_proc_fe = add_post_proc_fe();
1947
1948 auto set_time_monitor = [&](auto dm, auto ts) {
1949     auto monitor_ptr = boost::make_shared<FEMethod>();
1950
1951     monitor_ptr->preProcessHook = []() { return 0; };
1952     monitor_ptr->operatorHook = []() { return 0; };
1953     monitor_ptr->postProcessHook = [&]() {
1954         MoFEMFunctionBegin;
1955
1956         if (!post_proc_fe)
1957             SETERRQ(PETSC_COMM_WORLD, MOFEM_DATA_INCONSISTENCY,
1958                     "Null pointer for post proc element");
1959
1960         CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
1961                                         post_proc_fe);
1962         CHKERR post_proc_fe->writeFile(
1963             "level_set_" +
1964             boost::lexical_cast<std::string>(monitor_ptr->ts_step) + ".h5m");
1965         MoFEMFunctionReturn(0);
1966     };
1967
1968     boost::shared_ptr<FEMethod> null;
1969     DMMoFEMTSSetMonitor(sub_dm, ts, simple->getDomainFEName(), monitor_ptr,
1970                         null, null);
1971
1972     return monitor_ptr;
1973 };
1974
1975 auto ts = pip->createTSIM(sub_dm);
1976

```

```

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     };

```

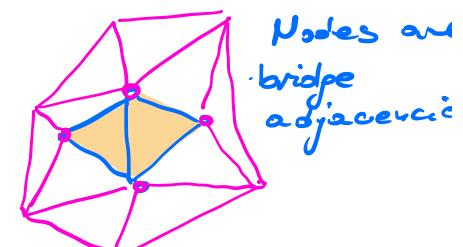
- ① Calculate error.
 ② Project error to paraview
 ③ Refine mesh.
 ④ Reset time solver

```

2065     CHKERR TSSetIJacobian(ts, B, B, TsSetIJacobian, ts_ctx);
2066     MoFEMFunctionReturn(0);
2067 }
2068
2069     CHKERR refine_and_project(ts);
2070     CHKERR ts_reset_theta(ts);
2071
2072     MoFEMFunctionReturn(0);
2073 }
2074
2075     auto ts_post_step = [] (TS ts) { return 0; };
2076
2077     CHKERR TSSetPreStep(ts, ts_pre_step);
2078     CHKERR TSSetPostStep(ts, ts_post_step);
2079
2080     CHKERR TSSolve(ts, NULL);
2081
2082     MoFEMFunctionReturn(0);
2083 }
2084
2085 MoFEMErrorCode LevelSet::refineMesh(WrapperClass &&wp) {
2086     MoFEMFunctionBegin;
2087
2088     auto simple = mField.getInterface<Simple>();
2089     auto bit_mng = mField.getInterface<BitRefManager>();
2090     ParallelComm *pcomm =
2091         ParallelComm::get_pcomm(&mField.get_moab(), MYPCOMM_INDEX);
2092     auto proc_str = boost::lexical_cast<std::string>(mField.get_comm_rank());
2093
2094     auto set_bit = [] (auto l) { return BitRefLevel().set(l); };
2095
2096     auto save_range = [&] (const std::string name, const Range &r) {
2097         MoFEMFunctionBegin;
2098         auto meshset_ptr = get_temp_meshset_ptr(mField.get_moab());
2099         CHKERR mField.get_moab().add_entities(*meshset_ptr, r);
2100         CHKERR mField.get_moab().write_file(name.c_str(), "VTK", "", meshset_ptr->get_ptr(), 1);
2101
2102         MoFEMFunctionReturn(0);
2103     };
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     };

```

Select elements
for refinement



```

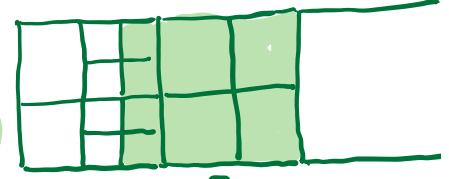
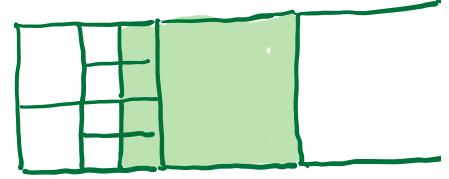
2151     ents = get_neighbours_by_bridge_vertices(ents);
2152
2153 #ifndef NDEBUG
2154     if (debug) {
2155         auto meshset_ptr = get_temp_meshset_ptr(mField.get_moab());
2156         CHK_MOAB_THROW(mField.get_moab().add_entities(*meshset_ptr, ents),
2157                         "add entities to meshset");
2158         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
2195         Range skin_edges;
2196         CHKERR skin.find_skin(0, level_ll, false, skin_edges);
2197         Range skin_parents;
2198         for (auto lll = 0; lll <= ll; ++lll) {
2199             CHKERR bit_mng->updateRangeByParent(skin_edges, skin_parents);
2200             skin_edges = skin_parents;
2201         }
2202         BitRefLevel bad_bit;
2203         for (auto lll = 0; lll <= ll - 2; ++lll) {
2204             bad_bit[lll] = true;
2205         }
2206         CHKERR bit_mng->filterEntitiesByRefLevel(bad_bit, BitRefLevel().set(),
2207                                         skin_edges);
2208
2209         Range skin_adj_ents;
2210         CHKERR mField.get_moab().get_adjacencies(skin_parents, SPACE_DIM, false,
2211                                         skin_adj_ents,
2212                                         moab::Interface::UNION);
2213         CHKERR bit_mng->filterEntitiesByRefLevel(bad_bit, BitRefLevel().set(),
2214                                         skin_adj_ents);
2215         skin_adj_ents = intersect(skin_adj_ents, level_ents);
2216         if (!skin_adj_ents.empty()) {
2217             level_ents = subtract(level_ents, skin_adj_ents);
2218             Range skin_adj_ents_children;
2219             bit_mng->updateRangeByChildren(skin_adj_ents, skin_adj_ents_children);
2220             level_ents.merge(skin_adj_ents_children);
2221         }
2222         MoFEMFunctionReturn(0);
2223     };
2224
2225     CHKERR fix_neighbour_level(l);
2226
2227     CHKERR mField.getInterface<CommInterface>()->synchroniseEntities(
2228         level_ents);
2229
2229     // get lower dimension entities for level "l"
2230     for (auto d = 0; d != SPACE_DIM; ++d) {
2231         if (d == 0) {
2232             CHKERR mField.get_moab().get_connectivity(
2233                 level_ents.subset_by_dimension(SPACE_DIM), level_ents, true);
2234         } else {
2235             CHKERR mField.get_moab().get_adjacencies(

```

* Get entities of previous refinement

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

Takes children of elements to refine and add them to elements of new refined mesh



this code make 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(),

```

✓ set bits to mark new refined mesh

Set bit indicate skeleton element ie. elements or current mesh or their parents.

```

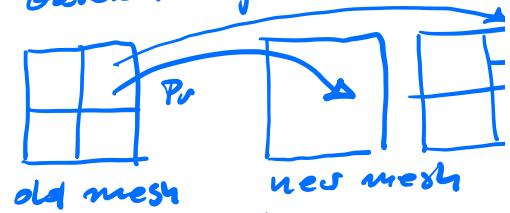
2323 CHKERR bit_mng->setNthBitRefLevel(level0, current_bit, true);
2324 CHKERR bit_mng->setNthBitRefLevel(level0, aggregate_bit, true);
2325 CHKERR bit_mng->setNthBitRefLevel(level0, skeleton_bit, true);
2326
2327 CHKERR wp.setBits(*this, 0);
2328 CHKERR wp.runCalcs(*this, 0);
2329 for (auto l = 0; l != 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
2372   Range prj_ents;
2373   CHKERR bit_mng->getEntitiesByDimAndRefLevel(BitRefLevel().set(projection_bit),
2374                                                 BitRefLevel().set(), SPACE_DIM,
2375                                                 prj_ents);
2376
2377   for (auto l = 0; l != nb_levels; ++l) {
2378     CHKERR bit_mng->updateRangeByParent(prj_ents, prj_ents);
2379   }
2380   current_ents = subtract(current_ents, prj_ents);
2381
2382   auto test_mesh_bit = [&](FEMethod *fe_ptr) {
2383     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
2384       current_bit);
2385   };
2386   auto test_prj_bit = [&](FEMethod *fe_ptr) {
2387     return fe_ptr->numeredEntFiniteElementPtr->getBitRefLevel().test(
2388       projection_bit);
2389   };
2390   auto test_current_bit = [&](FEMethod *fe_ptr) {
2391     return current_ents.find(fe_ptr->getFEEntityHandle()) != current_ents.end();
2392   };
2393
2394   lhs_fe->exeTestHook = test_mesh_bit;
2395   rhs_fe_prj->exeTestHook = test_prj_bit;
2396   rhs_fe_current->exeTestHook = test_current_bit;
2397
2398   BitRefLevel remove_mask = BitRefLevel().set(current_bit);
2399   remove_mask.flip(); // DOFs which are not on bit_domain_ele should be removed
2400   CHKERR prb_mng->removeDofsOnEntities(
2401     "DG_PROJECTION", "L", BitRefLevel().set(), remove_mask, nullptr, 0,
2402     MAX_DOFs_ON_ENTITY, 0, 100, NOISY, true);
2403
2404   CHKERR AddHOOps<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
2405     lhs_fe->getOpPtrVector(), {potential_velocity_space, L2});
2406   lhs_fe->getOpPtrVector().push_back(new OpMassLL("L", "L"));
2407
2408   auto l_vec = boost::make_shared<VectorDouble>();

```

Run "n" refinement steps.

Project solution from previous refinement & tri step, to new mesh at refinement.

This is discontinuous
galerkin projection



New element is finer (prolongation)

& new element is coarser

(restriction)

```

2409 CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
2410     rhs_fe_prj->getOpPtrVector(), {potential_velocity_space, L2});
2411 rhs_fe_prj->getOpPtrVector().push_back(
2412     new OpCalculateScalarFieldValues("L", l_vec));
2413 auto get_parent_this = [&]() {
2414     auto fe_parent_this = boost::make_shared<DomianParentEle>(mField);
2415     fe_parent_this->getOpPtrVector().push_back(
2416         new OpScalarFieldL("L", l_vec));
2417     return fe_parent_this;
2418 };
2419
2420 auto get_parents_fe_ptr = [&](auto this_fe_ptr) {
2421     std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
2422     for (int l = 0; l <= nb_levels; ++l)
2423         parents_elems_ptr_vec.emplace_back(
2424             boost::make_shared<DomianParentEle>(mField));
2425     for (auto l = 1; l <= nb_levels; ++l) {
2426         parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
2427             new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
2428                             BitRefLevel().set(current_bit).flip(), this_fe_ptr,
2429                             BitRefLevel().set(current_bit),
2430                             BitRefLevel().set()));
2431     }
2432     return parents_elems_ptr_vec[0];
2433 };
2434
2435 auto this_fe_ptr = get_parent_this();
2436 auto parent_fe_ptr = get_parents_fe_ptr(this_fe_ptr);
2437 rhs_fe_prj->getOpPtrVector().push_back(
2438     new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
2439                     BitRefLevel().set(current_bit).flip(), this_fe_ptr,
2440                     BitRefLevel().set(current_bit), BitRefLevel().set()));
2441 };
2442
2443 auto set_prj_from_parent = [&](auto rhs_fe_current) {
2444     auto get_parent_this = [&]() {
2445         auto fe_parent_this = boost::make_shared<DomianParentEle>(mField);
2446         fe_parent_this->getOpPtrVector().push_back(
2447             new OpCalculateScalarFieldValues("L", l_vec));
2448         return fe_parent_this;
2449     };
2450
2451     auto get_parents_fe_ptr = [&](auto this_fe_ptr) {
2452         std::vector<boost::shared_ptr<DomianParentEle>> parents_elems_ptr_vec;
2453         for (int l = 0; l <= nb_levels; ++l)
2454             parents_elems_ptr_vec.emplace_back(
2455                 boost::make_shared<DomianParentEle>(mField));
2456         for (auto l = 1; l <= nb_levels; ++l) {
2457             parents_elems_ptr_vec[l - 1]->getOpPtrVector().push_back(
2458                 new OpRunParent(parents_elems_ptr_vec[l], BitRefLevel().set(),
2459                                 BitRefLevel().set(projection_bit).flip(),
2460                                 this_fe_ptr, BitRefLevel().set(projection_bit),
2461                                 BitRefLevel().set()));
2462         }
2463         return parents_elems_ptr_vec[0];
2464     };
2465
2466     auto this_fe_ptr = get_parent_this();
2467     auto parent_fe_ptr = get_parents_fe_ptr(this_fe_ptr);
2468
2469     auto reset_op_ptr = new DomainEleOp(NOSPACE, DomainEleOp::OPSPACE);
2470     reset_op_ptr->doWorkRhsHook = [&](DataOperator *op_ptr, int, EntityType,
2471                                         EntData &) {
2472         l_vec->resize(static_cast<DomainEleOp *>(op_ptr)->getGaussPts().size2(),
2473                         false);
2474         l_vec->clear();
2475         return 0;
2476     };
2477     rhs_fe_current->getOpPtrVector().push_back(reset_op_ptr);
2478     rhs_fe_current->getOpPtrVector().push_back(
2479         new OpRunParent(parent_fe_ptr, BitRefLevel().set(),
2480                         BitRefLevel().set(projection_bit).flip(), this_fe_ptr,
2481                         BitRefLevel(), BitRefLevel()));
2482     rhs_fe_current->getOpPtrVector().push_back(new OpScalarFieldL("L", l_vec));
2483 };
2484
2485 set_prj_from_child(rhs_fe_prj);
2486 set_prj_from_parent(rhs_fe_current);
2487
2488 boost::shared_ptr<FEMethod> null_fe;
2489 smartGetDMKspCtx(sub_dm)->clearLoops();
2490 CHKERR DMMoFEMKSPSetComputeOperators(sub_dm, simple->getDomainFEName(),
2491                                         lhs_fe, null_fe, null_fe);
2492 CHKERR DMMoFEMKSPSetComputeRHS(sub_dm, simple->getDomainFEName(), rhs_fe_prj,

```

```

2495           rhs_fe_current, null_fe, null_fe);
2496     auto ksp = MoFEM::createKSP(mField.get_comm());
2497     CHKERR KSPSetDM(ksp, sub_dm);
2498 
2499     CHKERR KSPSetDM(ksp, sub_dm);
2500     CHKERR KSPSetFromOptions(ksp);
2501     CHKERR KSPSetUp(ksp);
2502 
2503     auto L = smartCreateDMVector(sub_dm);
2504     auto F = smartVectorDuplicate(L);
2505 
2506     CHKERR KSPSolve(ksp, F, L);
2507     CHKERR VecGhostUpdateBegin(L, INSERT_VALUES, SCATTER_FORWARD);
2508     CHKERR VecGhostUpdateEnd(L, INSERT_VALUES, SCATTER_FORWARD);
2509     CHKERR DMoFEMMeshToLocalVector(sub_dm, L, INSERT_VALUES, SCATTER_REVERSE);
2510 
2511     auto [error, th_error] = evaluateError();
2512     MOFEM_LOG("LevelSet", Sev::inform) << "Error indicator " << error;
2513 
2514     auto post_proc = [&](auto dm, auto out_name, auto th_error) {
2515       MoFEMFunctionBegin;
2516       auto post_proc_fe =
2517         boost::make_shared<PostProcBrokenMeshInMoab<DomainEle>>(mField);
2518       post_proc_fe->setTagsToTransfer({th_error});
2519       post_proc_fe->exeTestHook = test_mesh_bit;
2520 
2521       using OpPPMap = OpPostProcMapInMoab<SPACE_DIM, SPACE_DIM>;
2522 
2523       auto l_vec = boost::make_shared<VectorDouble>();
2524       auto l_grad_mat = boost::make_shared<MatrixDouble>();
2525       CHKERR AddH0Ops<SPACE_DIM, SPACE_DIM, SPACE_DIM>::add(
2526         post_proc_fe->getOpPtrVector(), {potential_velocity_space, L2});
2527       post_proc_fe->getOpPtrVector().push_back(
2528         new OpCalculateScalarFieldValues("L", l_vec));
2529       post_proc_fe->getOpPtrVector().push_back(
2530         new OpCalculateScalarFieldGradient<SPACE_DIM>("L", l_grad_mat));
2531 
2532       post_proc_fe->getOpPtrVector().push_back(
2533 
2534         new OpPPMap(
2535 
2536           post_proc_fe->getPostProcMesh(), post_proc_fe->getMapGaussPts(),
2537 
2538             {{"L", l_vec}},
2539 
2540             {{"GradL", l_grad_mat}},
2541 
2542             {}, {}
2543 
2544         );
2545 
2546       CHKERR DMoFEMLoopFiniteElements(dm, simple->getDomainFEName(),
2547                                         post_proc_fe);
2548       post_proc_fe->writeFile(out_name);
2549       MoFEMFunctionReturn(0);
2550     };
2551 
2552     if constexpr (debug)
2553       CHKERR post_proc(sub_dm, "dg_projection.h5m", th_error);
2554 
2555     MoFEMFunctionReturn(0);
2556   }

```

Note that field "L" i.e. $\varphi \in L^2(\Omega)$, thus projection can be performed element by element.

Here we use generic KSP solver and preconditioner. By implementing problem tailored preconditioner using Cholesky solver element by element optimization can be implemented.