Local Grid Refinement in Corner-point Grids

Antonella Ritorto

OPM-OP AS

antonella.ritorto@opm-op.com

OPM Summit 2024

April 9-10, Oslo

Local **G**rid **R**efinement

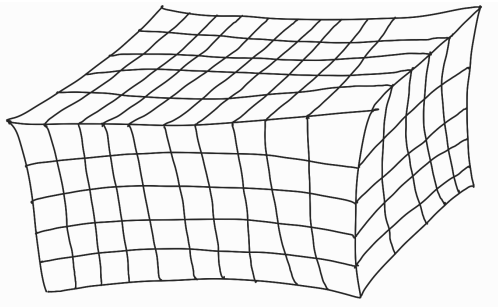
Local Grid Refinement in CpGrid

Cooperation between:

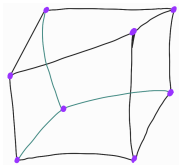
- Bård Skaflestad (**SINTEF**)
- Cintia Goncalves Machado (*ex*-**TNO**)
- Eduardo Barros (**TNO**)
- Markus Blatt (**OPM-OP**)
- Negar Khoshnevis (**TNO**)

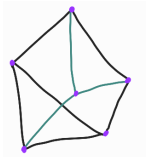
Quick glance at CpGrid

Corner point grid $\rightsquigarrow$ degenerated and distorted **Cartesian grid**

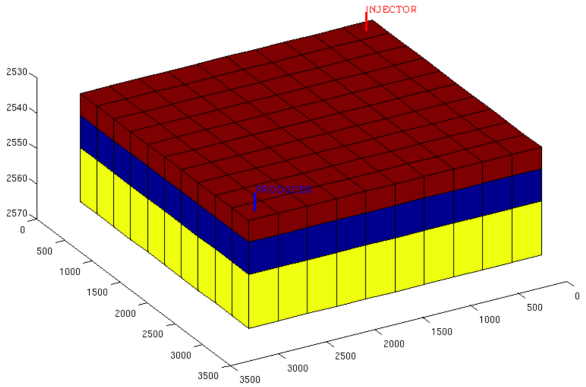


Degenerated/deformed cells

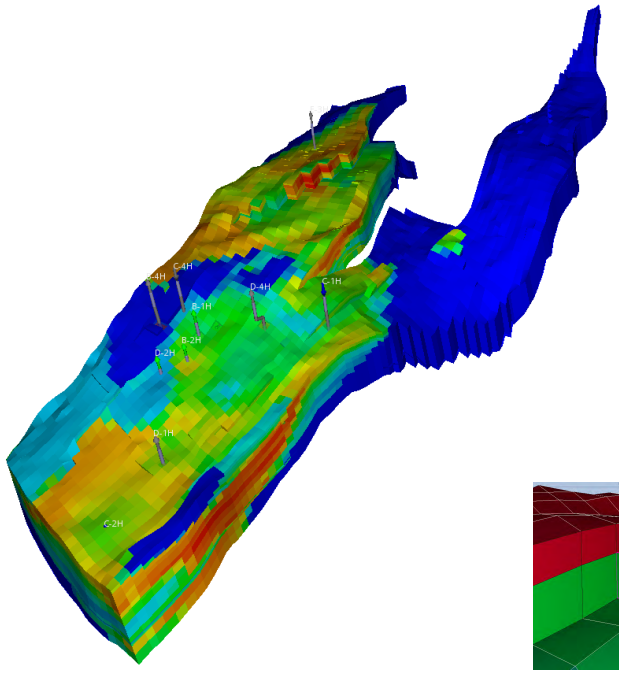


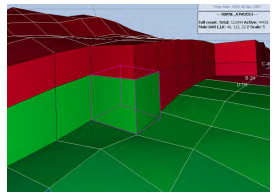


(Reg.) Cartesian CpGrid - SPE1



(Irreg.) CpGrid - Norne oil field





- Mapping from cells to the underlying **Cartesian Index**
- Each cell can be **ACTIVE** or **INACTIVE**

Goals

Project goals

- Support **simulation for CpGrid with multiple LGRs**
  - active cells
  - no wells in LGRs
  - disjoint LGRs
- Provide **output files**

Talk goals

- LGR evolution for CpGrid
  - How was it?
  - What has been done?
  - What's next?
- Intend to summarize **more than 68 merged PRs** in

  opm-grid
  opm-simulators
  opm-common
  opm-test

LGR evolution for CpGrids

How it was...

0. How it was (June 2022)

Refine a geometry object into a regular grid #582

Merged blattms merged 2 commits into OPM:master from verveerpj:refine-geometry on Jun 8, 2022

Conversation 34 Commits 2 Checks 0 Files changed 2

verveerpj commented on May 23, 2022 Member

This PR is initially for discussing on how proceed with local grid refinement, in particular with @blattms

This PR adds a method to refine a geometry object in regular grid, returning a vector of the new geometry objects.

Some remarks:

- Currently implemented using local coordinates, which seems natural, but could also be done more directly. In fact, the test uses a direct calculation to check the method.
- The geometry objects does not manage some of its storage, so that is currently passed as a vector of arrays for each newly created geometry object. Maybe that needs to be passed as contiguous memory for efficiency, but that does not seem to be necessary yet for the purpose of the current discussion.

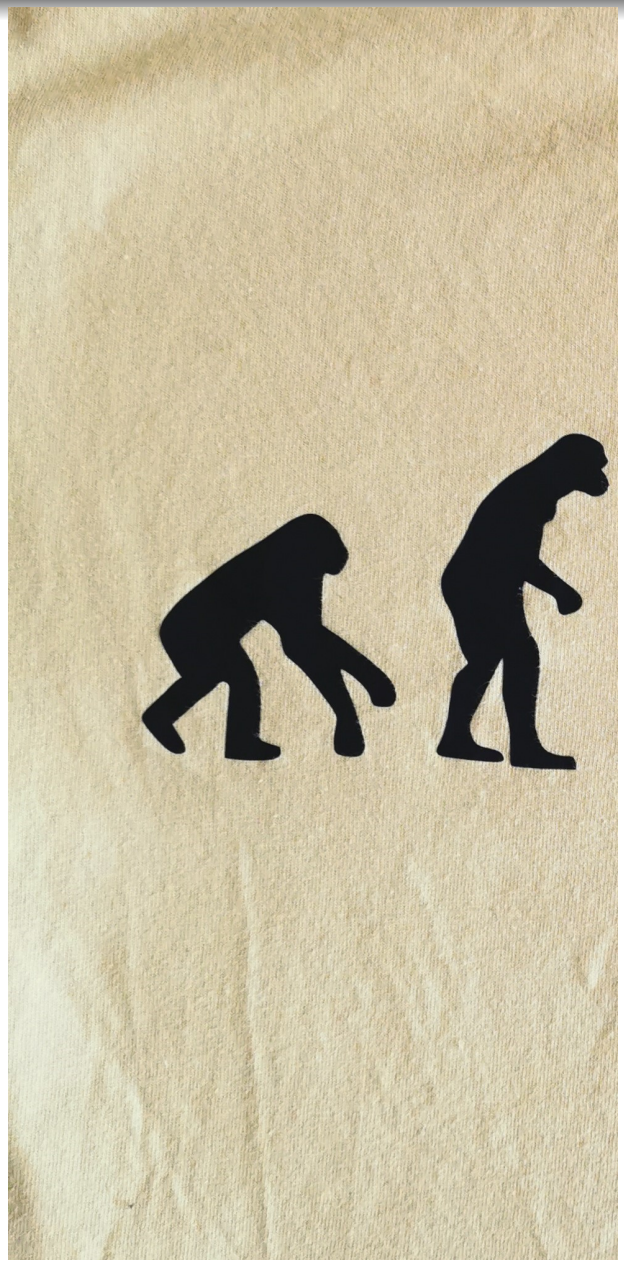
```
/// @brief Refine a single cell with regular intervals.
/// @param cells The number of sub-cells in each direction,
/// @param corner_storage A vector of mutable references to storage for the corners of each new cell.
/// @param indices_storage A vector of mutable references to storage for the indices of each new cell.
std::vector<Geometry<3, cdim>> refine(const std::array<int, 3>& cells_per_dim,
                                      std::vector<EntityVariable<Geometry<0, 3>, 3>>& corner_storage,
                                      std::vector<std::array<int, 8>>& indices_storage)
```

Geometry$\langle 3, 0 \rangle$ **vector** (*cells*)

- Geometrical aspects:
  - center
  - volume

What has been done...

1. Refine **single** cell (from October 2022)

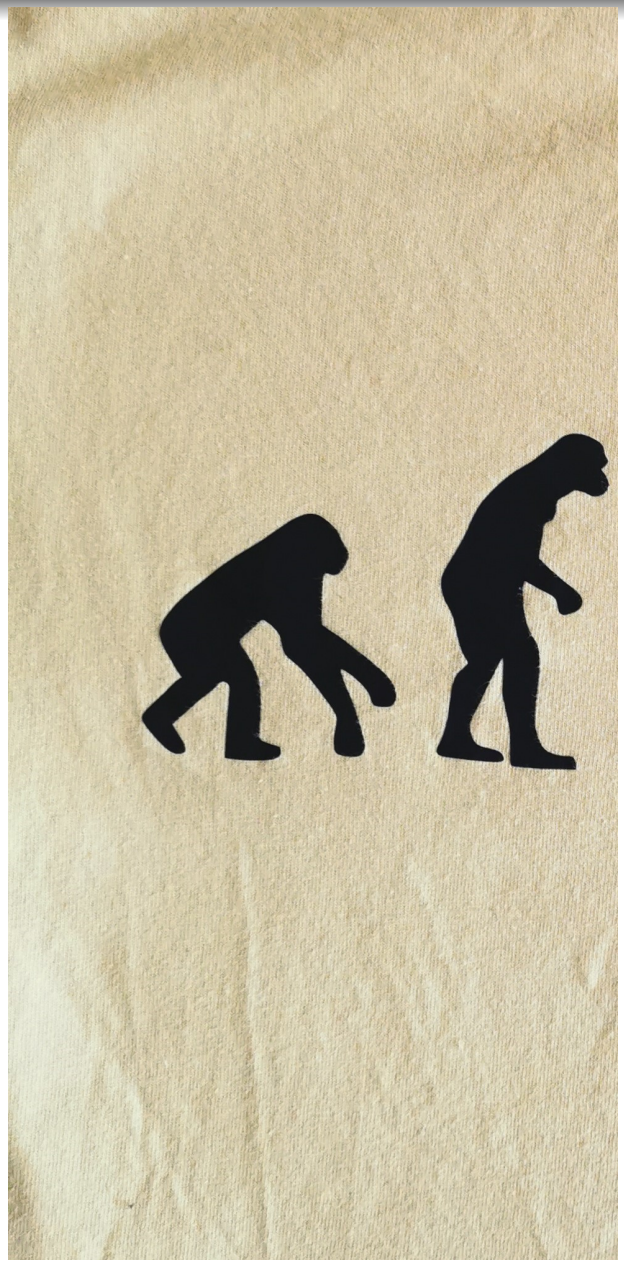


```
/**
 * @brief Refine a single cell with regular intervals.
 *
 * For each cell to be created, storage must be passed for its corners and the indices. That storage
 * must be externally managed, since the newly created geometry structures only store pointers and do
 * not free them on destruction.
 *
 * @param      cells_per_dim    The number of sub-cells in each direction,
 * @param[out] refined_geom     Geometry Policy for the refined geometries. Those will be added there.
 * @param[out] indices_storage  A vector of mutable references to storage for the indices of each new cell.
 * @return A vector with the created cells.
 * @todo We do not need to return anything here.
 */
void refine(const std::array<int, 3>& cells_per_dim,
            DefaultGeometryPolicy& all_geom,
            std::vector<std::array<int, 8>>&  global_refined_cell8corners_indices_storage)
```

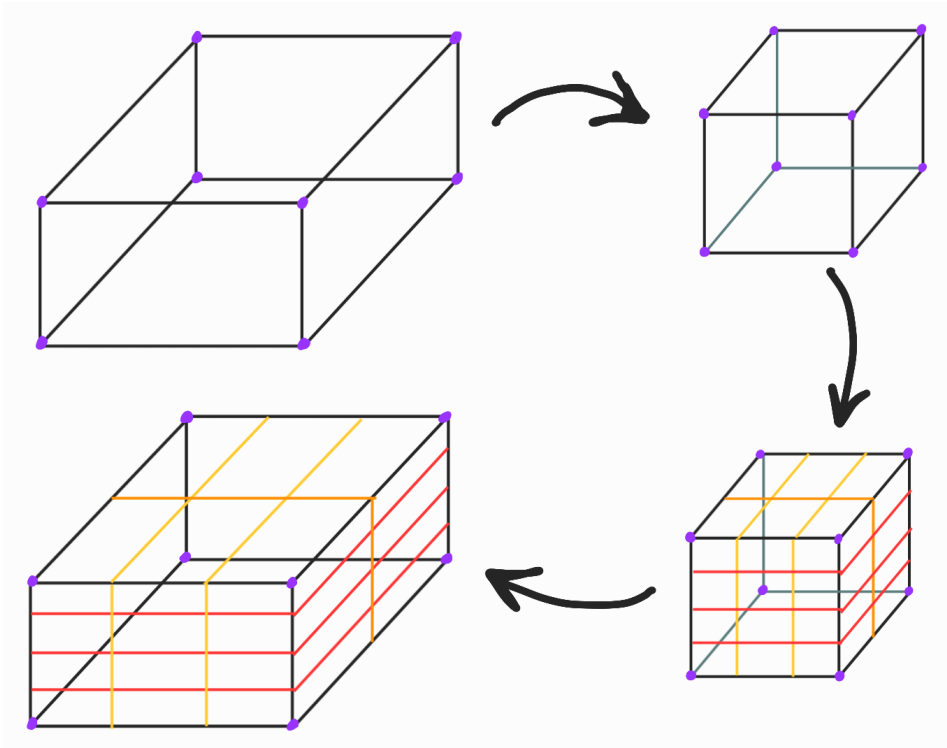
All geometries (*corners*, *faces*, *cells*)

- Geometrical aspects:
  - center
  - volume
- **Topological aspects**:
  - face to its 4 corners
  - face to its neighboring cells
  - cell to its 8 corners
  - cell to its 6 faces

1. Refine **single** cell



```
/**
 * @brief Refine a single cell with regular intervals.
 *
 * For each cell to be created, storage must be passed for its corners and the indices. That storage
 * must be externally managed, since the newly created geometry structures only store pointers and do
 * not free them on destruction.
 *
 * @param      cells_per_dim    The number of sub-cells in each direction,
 * @param[out] refined_geom     Geometry Policy for the refined geometries. Those will be added there.
 * @param[out] indices_storage  A vector of mutable references to storage for the indices of each new cell.
 * @return A vector with the created cells.
 * @todo We do not need to return anything here.
 */
void refine(const std::array<int, 3>& cells_per_dim,
            DefaultGeometryPolicy& all_geom,
            std::vector<std::array<int, 8>>&  global_refined_cell8corners_indices_storage)
```



1. Refine **single** cell

Main challenge

- Create/store **topology** aspects (*numbering matters!*)
  - corners **numbering** consistent with CpGrid ✓

Cell to point

```
// INDEX of the global refined cell associated with 'kji'.
int refined_cell_idx = (k*cells_per_dim[0]*cells_per_dim[1]) + (j*cells_per_dim[0]) +i;
// 1. CENTER of the global refined cell associated with 'kji' (Vol3.)
// Compute the center of the local refined unit/reference cube associated with 'kji'.
const LocalCoordinate& local_refined_cell_center = {
    (.5 + i)/cells_per_dim[0], (.5 + j)/cells_per_dim[1], (.5 + k)/cells_per_dim[2]};
// Obtain the global refined center with 'this->global(local_refined_cell_center)'.
// 2. VOLUME of the global refined 'kji' cell
double refined_cell_volume = 0.0; // (computed below!)
// 3. All Global refined corners ("refined_corners")
// 4. Indices of the 8 corners of the global refined cell associated with 'kji'.
std::array<int,8> cell8corners_indices = { //
    (j*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + (i*(cells_per_dim[2]+1)) +k, // fake '0' {0,0,0}
    (j*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + ((i+1)*(cells_per_dim[2]+1)) +k, // fake '1' {1,0,0}
    ((j+1)*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + (i*(cells_per_dim[2]+1)) +k, // fake '2' {0,1,0}
    ((j+1)*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + ((i+1)*(cells_per_dim[2]+1)) +k, // fake '3' {1,1,0}
    (j*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + (i*(cells_per_dim[2]+1)) +k+1, // fake '4' {0,0,1}
    (j*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + ((i+1)*(cells_per_dim[2]+1)) +k+1, // fake '5' {1,0,1}
    ((j+1)*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + (i*(cells_per_dim[2]+1)) +k+1, // fake '6' {0,1,1}
    ((j+1)*(cells_per_dim[0]+1)*(cells_per_dim[2]+1)) + ((i+1)*(cells_per_dim[2]+1)) +k+1 // fake '7' {1,1,1}
};
```

1. Refine **single** cell

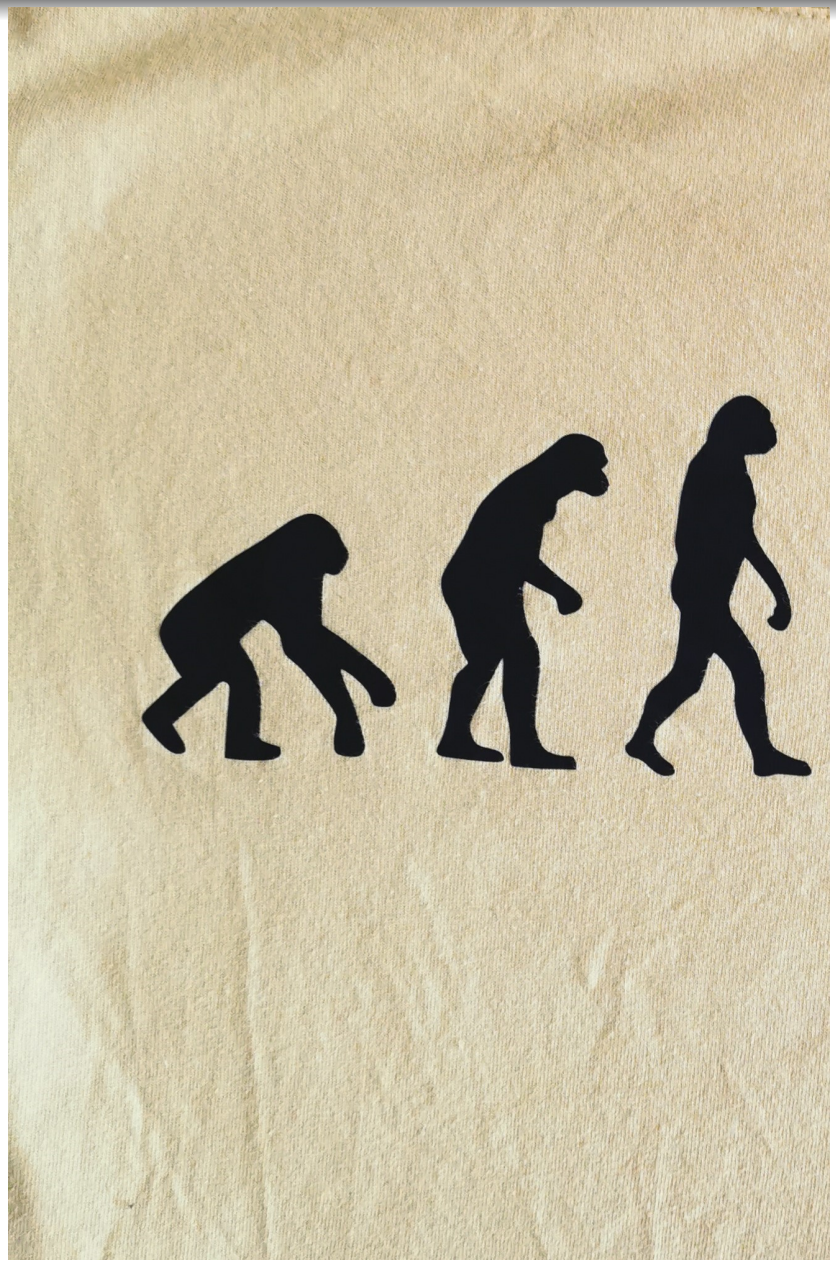
Main challenge

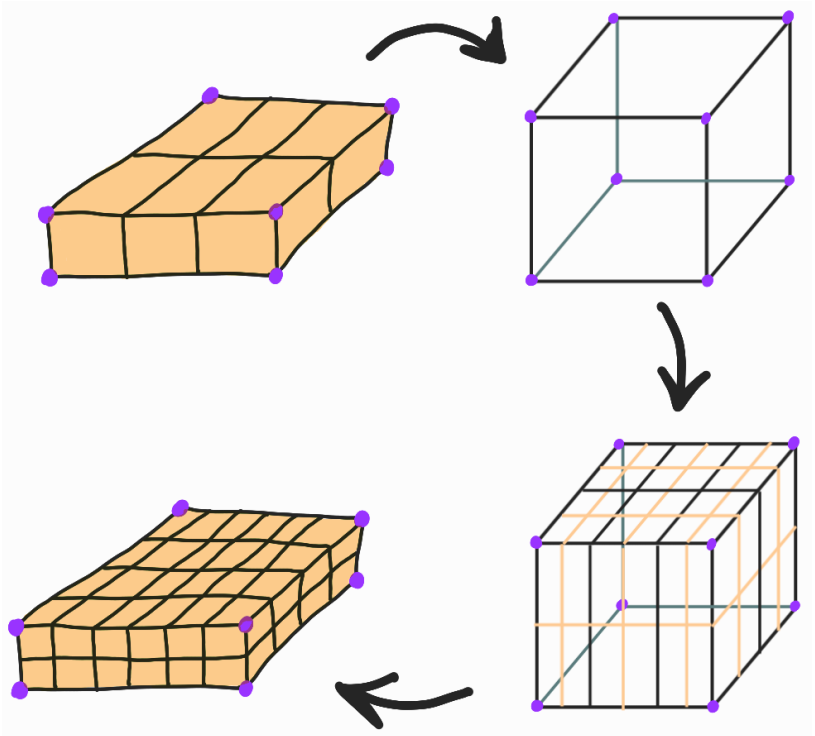
- Create/store **topology** aspects (*numbering matters!*)
  - corners **numbering** consistent with CpGrid ✓

Cell to face

```
// VOLUME HEXAHEDRON (GLOBAL REFINED 'CELL')
// Vol1. INDICES ('from 0 to 5') of the faces of the hexahedron (needed to acce
std::vector<int> hexa_to_face = { //hexa_face_0to5_indices = {
    // index face '0' bottom
    (k*cells_per_dim[0]*cells_per_dim[1]) + (j*cells_per_dim[0]) + i,
    // index face '1' front
    (cells_per_dim[0]*cells_per_dim[1]*(cells_per_dim[2]+1))
    +  ((cells_per_dim[0]+1)*cells_per_dim[1]*cells_per_dim[2])
    + (j*cells_per_dim[0]*cells_per_dim[2]) + (i*cells_per_dim[2]) + k,
    // index face '2' left
    (cells_per_dim[0]*cells_per_dim[1]*(cells_per_dim[2]+1))
    + (i*cells_per_dim[1]*cells_per_dim[2]) + (k*cells_per_dim[1]) + j,
    // index face '3' right
    (cells_per_dim[0]*cells_per_dim[1]*(cells_per_dim[2]+1))
    + ((i+1)*cells_per_dim[1]*cells_per_dim[2]) + (k*cells_per_dim[1]) + j,
    // index face '4' back
    (cells_per_dim[0]*cells_per_dim[1]*(cells_per_dim[2]+1)) +
    ((cells_per_dim[0]+1)*cells_per_dim[1]*cells_per_dim[2])
    + ((j+1)*cells_per_dim[0]*cells_per_dim[2]) + (i*cells_per_dim[2]) + k,
    // index face '5' top
    ((k+1)*cells_per_dim[0]*cells_per_dim[1]) + (j*cells_per_dim[0]) + i};
//
```

2. Refine **block** of cells





2. Refine **block** of cells

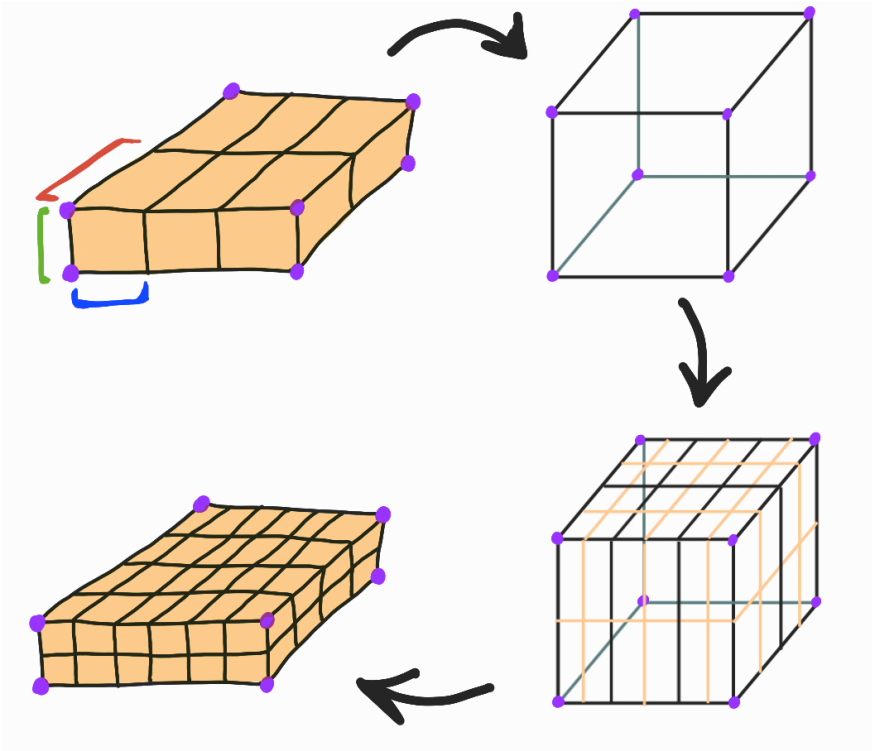
Main challenges

- Take into account

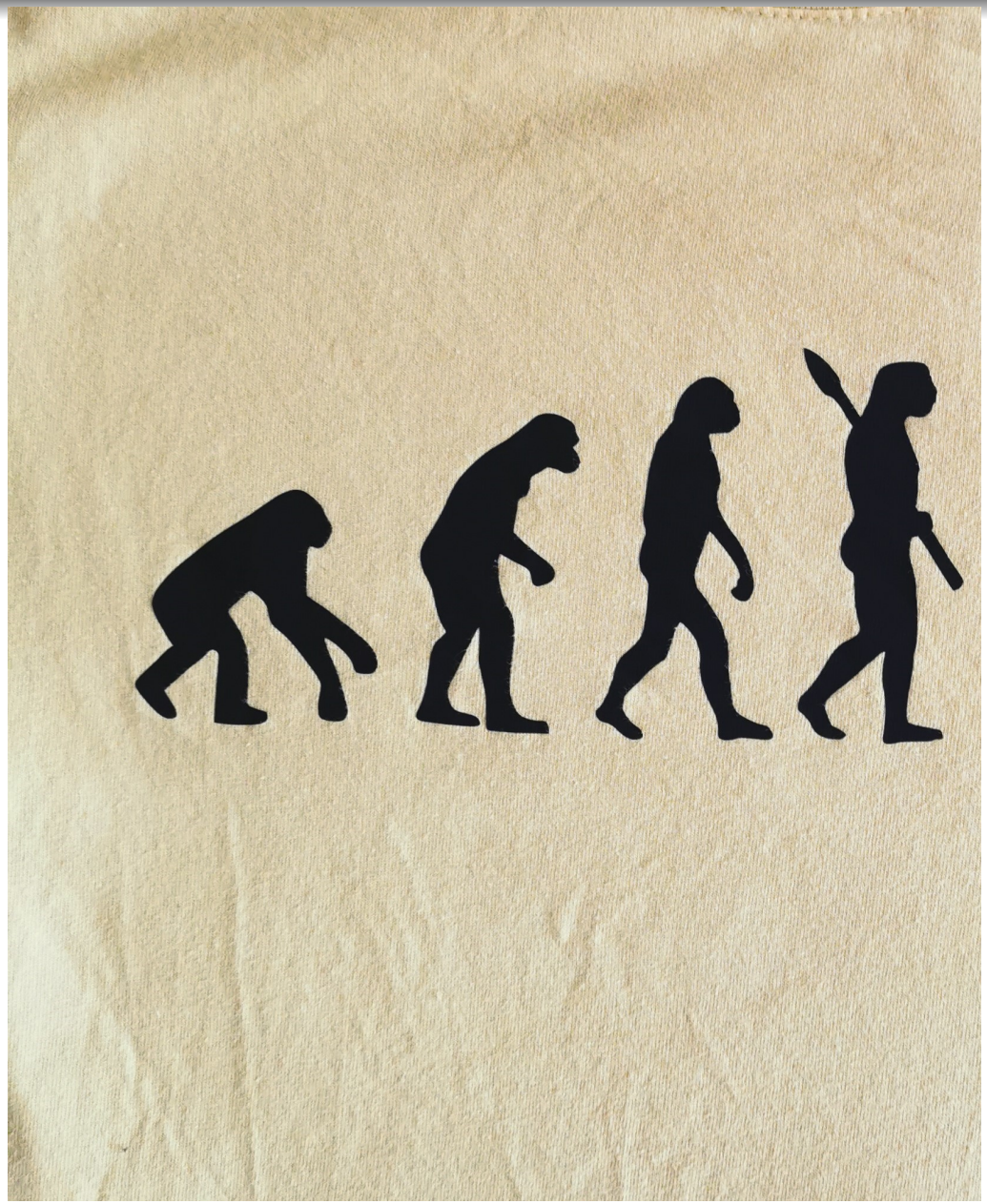
  widths
  lengths
  heights

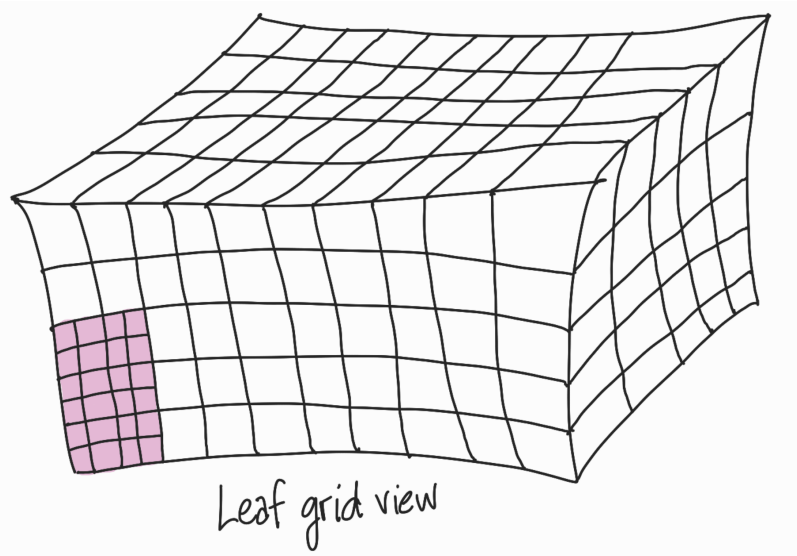
  of block cells

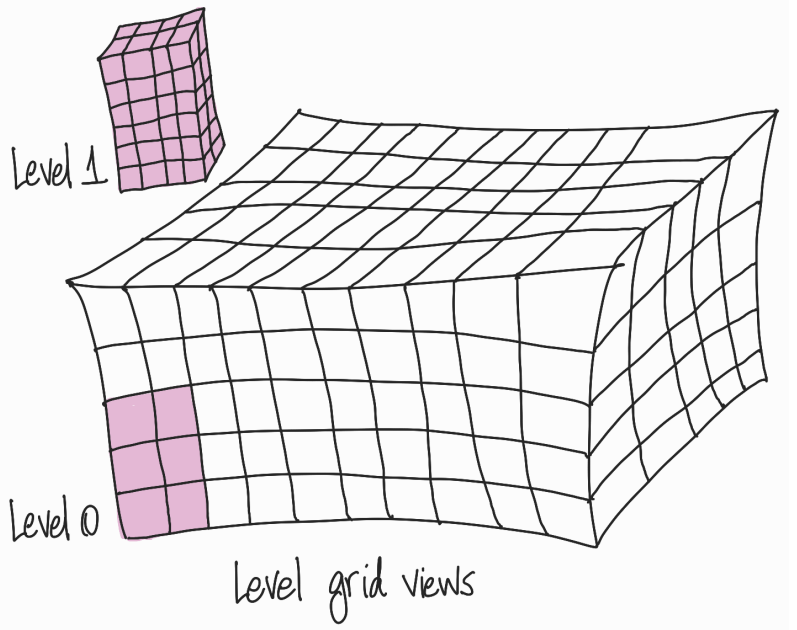
- Keep track on **parent-child** to create **topology** aspects (*numbering matters!*)



3. **One LGR** and its **updated leaf grid view**







3. **One LGR** and its **updated leaf grid view**

Main challenges

- Create/store a **level grid** in CpGrid

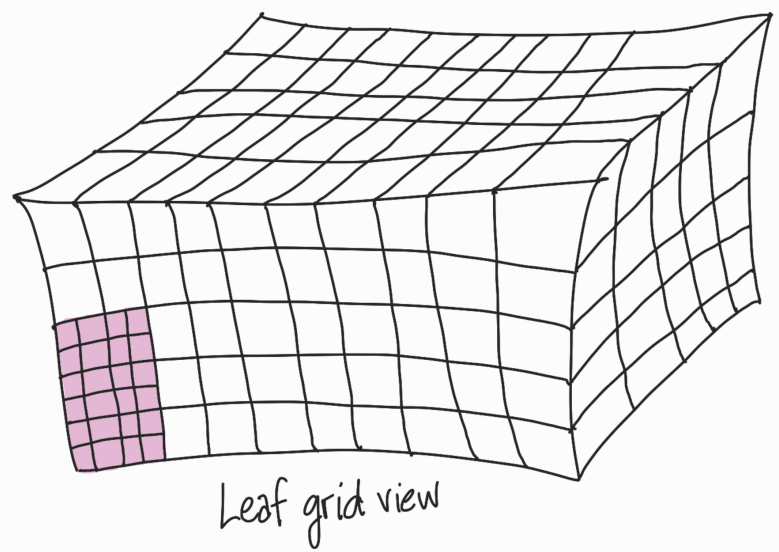
  Hierarchical grids

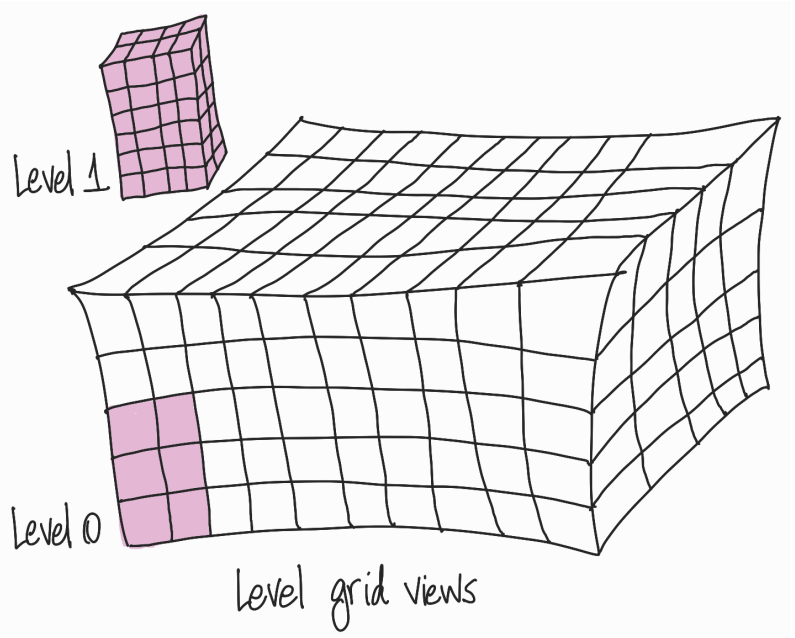
  LGR with its geometrical and topological features

- Identify **parent-child** relationships
  - Entity::father()
  - Entity::hasFather()
  - Entity::geometryInFather() ...

- Store/compute/iterate over **leaf grid view**

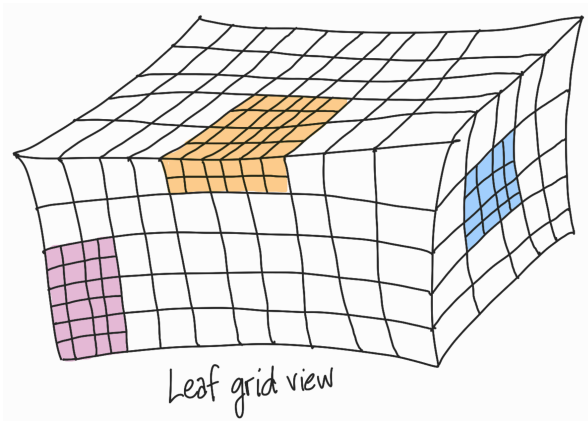
  Hierarchical grids and Leaf Grid View (iterators, ...)

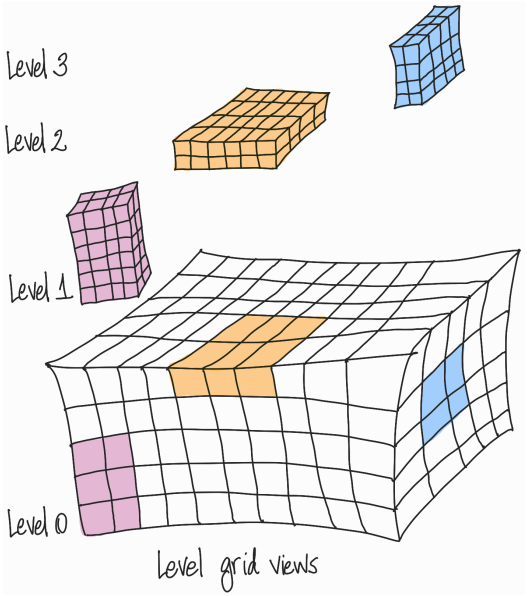




DUNE's adaptive grid interface

4. **Multiple LGRs** and its **updated leaf grid view**

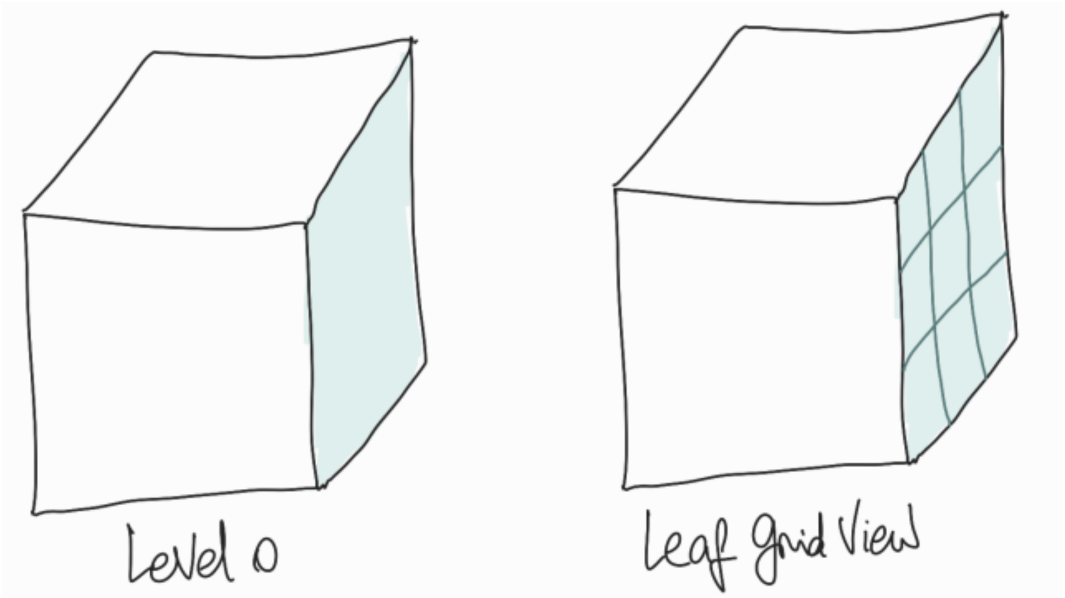




4. **Multiple LGRs** and its **updated leaf grid view**

Main challenges *(also present in one level)*

- Store on the **Leaf Grid View only once** each entity
- **Topology** aspects **on LGR boundary**



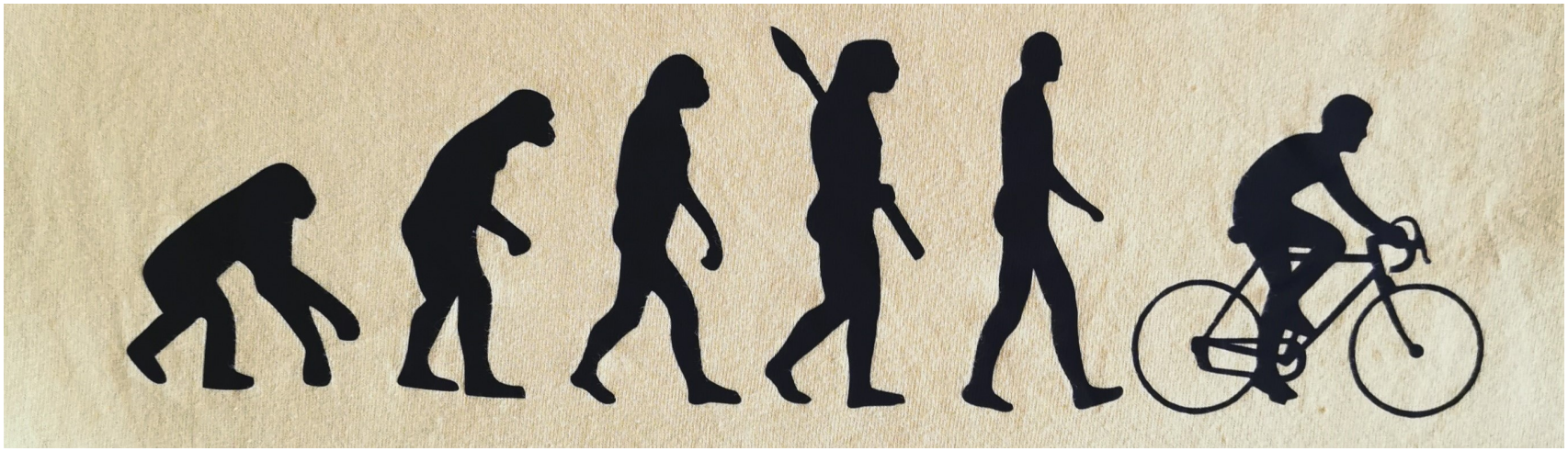
4. **Multiple LGRs** and its **updated leaf grid view**

```
// FACES COMING FROM LEVEL 0
for (int face = 0; face < static_cast<int>(this -> data_[0]->face_to_cell_.size()); ++face){
    if (level_to_leaf_faces[0][face] != -1){ // ONLY NEEDED FOR LEVEL 0
        const auto& leafFaceIdx =  level_to_leaf_faces[0][face];
        // Get the level data.
        const auto& level_data =  *(this->data_[0]);
        // Get the (face) entity (from level data).
        const auto& entity =  Dune::cpgrid::EntityRep<1>(face, true);
        // Get the face geometry.
        leaf_faces[leafFaceIdx] = (*(level_data.geometry_.geomVector(std::integral_constant<int,1>())))[entity];
        // Get the face tag.
        mutable_face_tags[leafFaceIdx] = level_data.face_tag_[entity];
        // Get the face normal.
        mutable_face_normals[leafFaceIdx] = level_data.face_normals_[entity];
        // Get old_face_to_point.
        auto old_face_to_point = level_data.face_to_point_[face];
        aux_face_to_point[leafFaceIdx].reserve(old_face_to_point.size());
        // Add the amount of points to the count num_points.
        num_points += old_face_to_point.size();
        for (int corn = 0; corn < 4; ++corn) {
            if (level_to_leaf_corners[0][old_face_to_point[corn]] == -1) {
                // In this case, the corner from level zero got replaced by a refined one.
                // Detect the corresponding LGR (if the corner appears in more than one LGR, then it's the last one)
                // and the refined corner index in that LGR, which is equivalen (meaning that both - corner
                // from level zero and refined corner - have exactly the same values in x-,y-, and z-coordinates.
                const auto [lgr, lgrCornIdx] = levelZeroToLGRsBoundaryCorners_oneToOne[{0, old_face_to_point[corn]}];
                aux_face_to_point[leafFaceIdx].push_back(level_to_leaf_corners[lgr][lgrCornIdx]);
            }
            else{// Corner not involded in any LGR.
                aux_face_to_point[leafFaceIdx].push_back(level_to_leaf_corners[0][old_face_to_point[corn]]);
            }
        } // end-corn-forloop
    } // end-if
} // end-face-for-loop
// FACES COMING FROM LGRS
```

4. **Multiple LGRs** and its **updated leaf grid view**

```
// Cell to point.
for (int corn = 0; corn < 8; ++corn) {
    // Auxiliary bool to identity boundary patch corners
    bool is_there_allPatchBoundCorn = false;
    for(const auto& [l0_oldCorner, level_newCorner] : levelZeroToLGRsBoundaryCorners_oneToOne) {
        is_there_allPatchBoundCorn = is_there_allPatchBoundCorn || (old_cell_to_point[corn] == l0_oldCorner[1]);
        if (is_there_allPatchBoundCorn) { //true-> coincides with one boundary patch corner
            leaf_cell_to_point[leafCellIdx][corn] = level_to_leaf_corners[level_newCorner[0]][level_newCorner[1]];
            break; // Go to the next corner (of the bondary of a patch)
        }
    }
    if(!is_there_allPatchBoundCorn) { // Corner does not belong to any patch boundary.
        leaf_cell_to_point[leafCellIdx][corn] = level_to_leaf_corners[0][old_cell_to_point[corn]];
    }
} // end-corn-forloop
// Cell to face.
for (const auto& face : old_cell_to_face) {   // Auxiliary bool to identity boundary patch faces
    bool is_there_allPatchBoundFace = false;
    for (const auto& [l0_boundFace, level_childrenList] : old_to_new_boundaryPatchFaces) {
        // l0_boundFace = {0,face},  level_childrenList = {patch +1, children_list}
        is_there_allPatchBoundFace = is_there_allPatchBoundFace || (face.index() == l0_boundFace[1]);
        //true-> coincides with one boundary patch face
        if (is_there_allPatchBoundFace) { // Face belongs to one of the patch boundaries.
            const auto levelTouched = std::get<0>(level_childrenList);
            for (const auto& new_face : std::get<1>(level_childrenList)) {
                //  level_to_leaf_faces[level][face] = leaf_face_index
                aux_cell_to_face.push_back({level_to_leaf_faces[levelTouched][new_face], face.orientation()});
            }
            is_there_allPatchBoundFace = true;
            break; // Go to the next face (on the boundary of a patch)
        }
    }
    if (!is_there_allPatchBoundFace) { // Face does not belong to any of the patch boundaries.
        aux_cell_to_face.push_back({level_to_leaf_faces[0][face.index()], face.orientation()});
    }
} // end-old_cell_to_face-forloop
```

5. Simulation



Simulation (partially) supported for CpGrid with LGRs #5218

Merged blattms merged 1 commit into OPM:master from aritorto:aboutLgrTrans 4 minutes ago

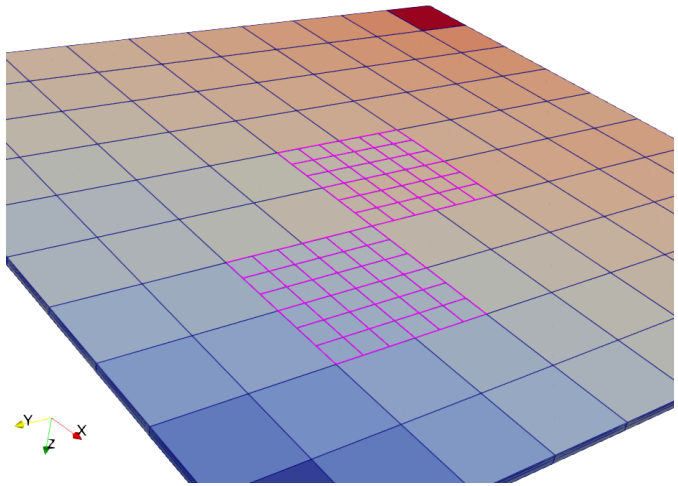
Conversation 3 Commits 1 Checks 0 Files changed 4

aritorto commented last month • edited Member

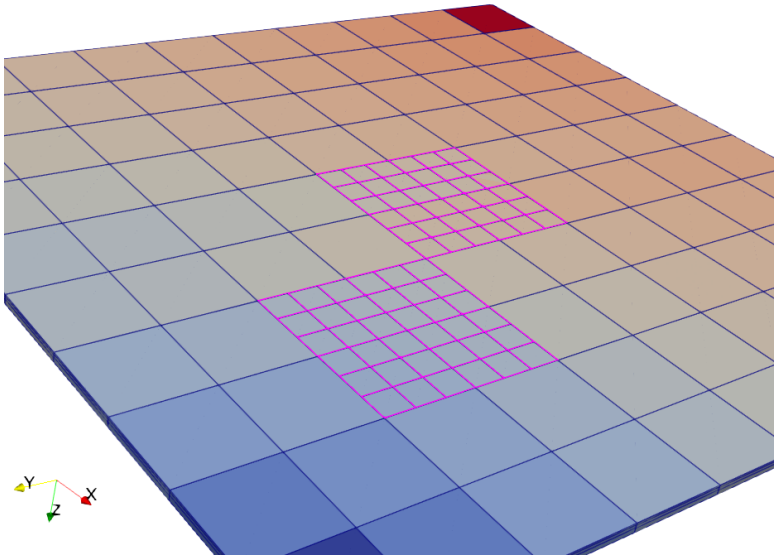
Simulation (partially) supported for CpGrid with multiple LGRs (local grid refinement). The assumptions are:

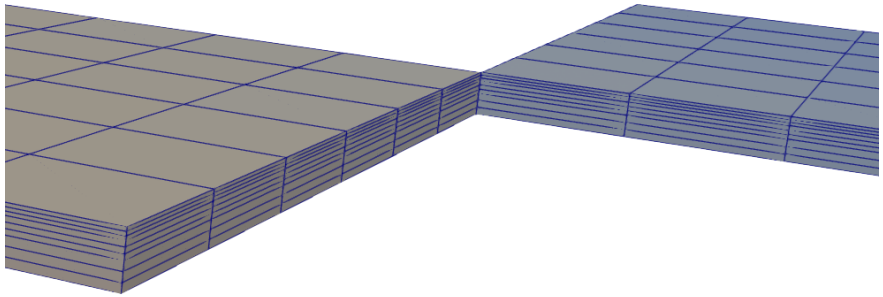
- disjoint LGRs,
- all the cells have to active,
- each set of cells to be refined has cuboid shape.

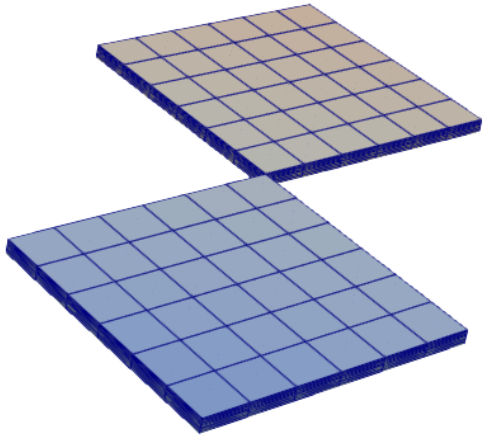
Future work: Output files.

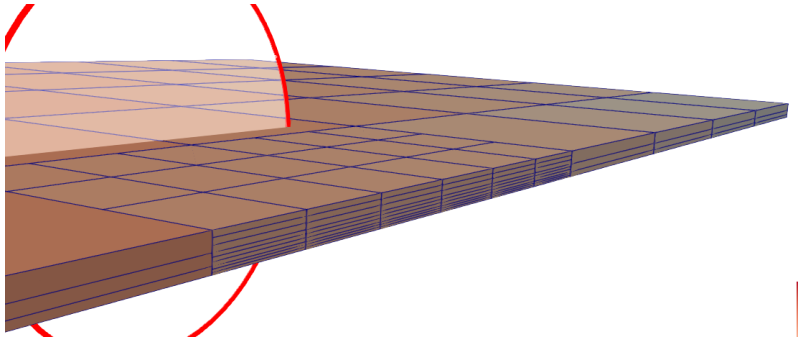


5. Simulation









- CARFIN is (partially) supported for CpGrid

```
GRID

CARFIN
-- NAME I1-I2 J1-J2 K1-K2 NX NY NZ
'LGR1'  5  6  5  6  1  3  6  6  9 /
ENDFIN

CARFIN
-- NAME I1-I2 J1-J2 K1-K2 NX NY NZ
'LGR2'  7  8  7  8  1  3  6  6  9 /
ENDFIN
```

5. Simulation

Main challenges

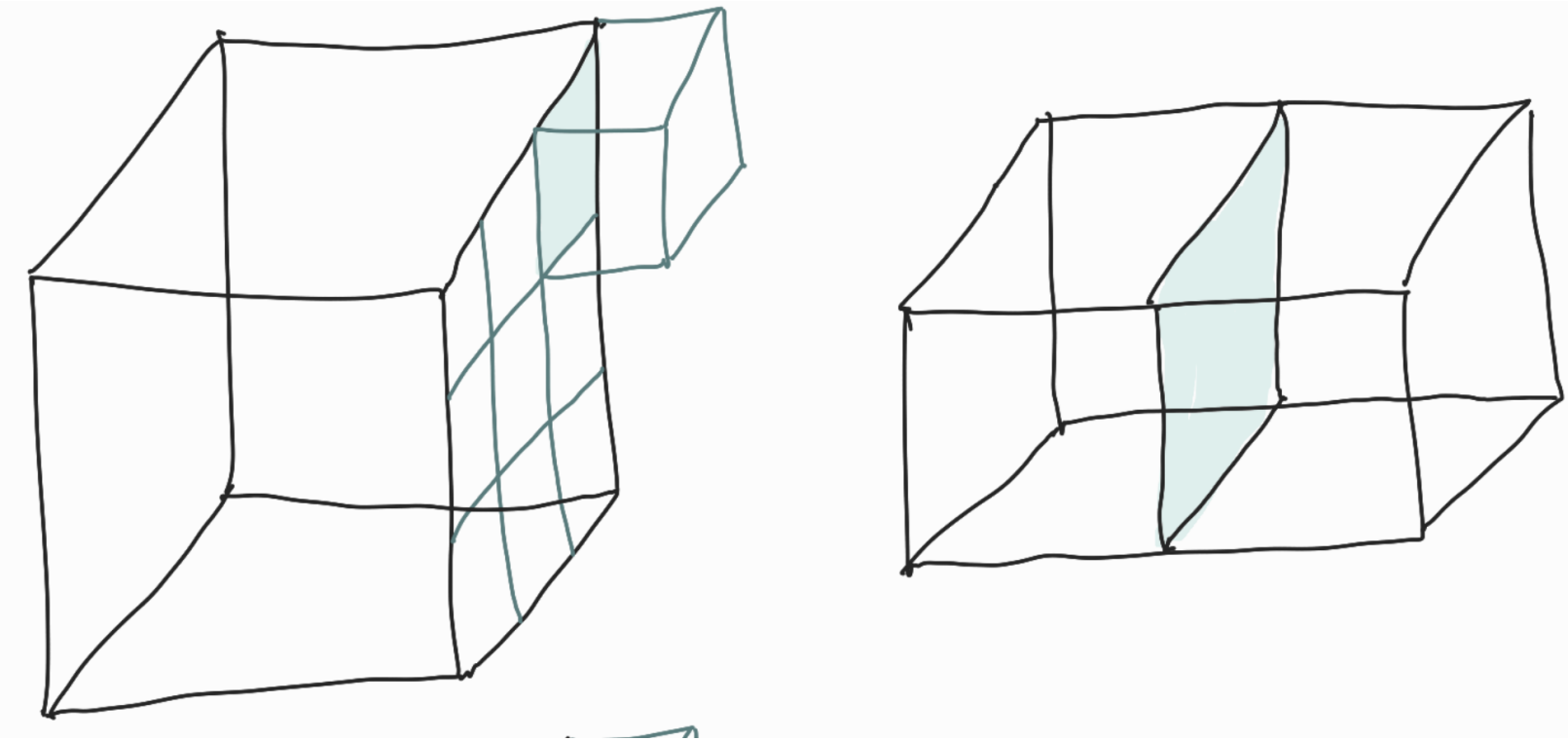
- Extensive and deep analysis in opm-simulators ebos files
  - **Cartesian index** (not unique!)
- **Assign field properties** to all leaf grid view elements

  NTG, (relative) permeability, dispersion, thermal props, material props, HystParams, **transmissibilities**, poro volume, ...
  - Search data via **element index** or **Cartesian index**
- Refactor **element centroids lookup**
- Refactor face centroids and **track parent intersection**
  - **Two-point flux approximation** related
- Connect LGRs from opm-common with LGRs from opm-grid
- ...

5. Simulation

- **Jumps** on the simulation on **LGR boundaries**

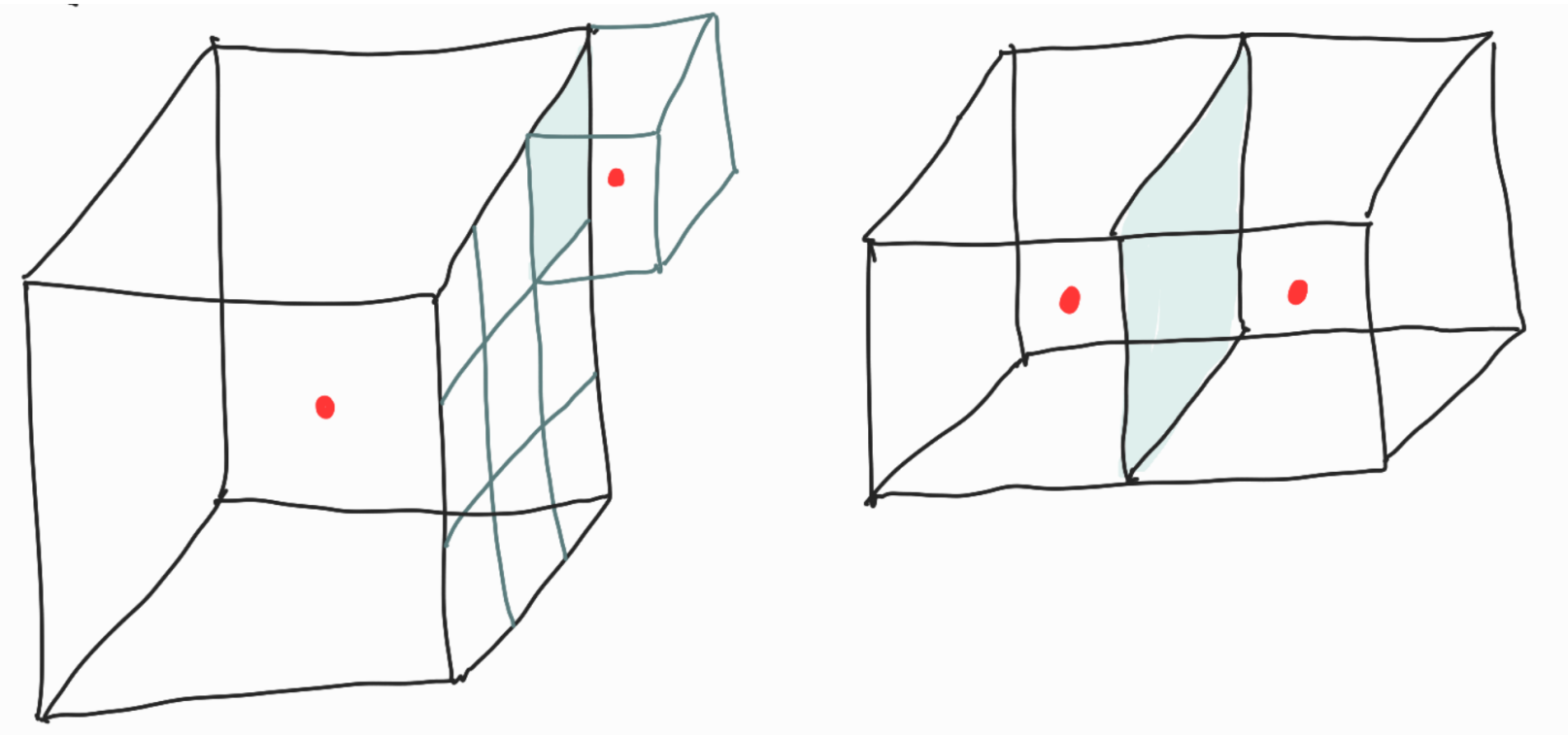
  Why? **Two-point flux approximation**



5. Simulation

- **Jumps** on the simulation on **LGR boundaries**

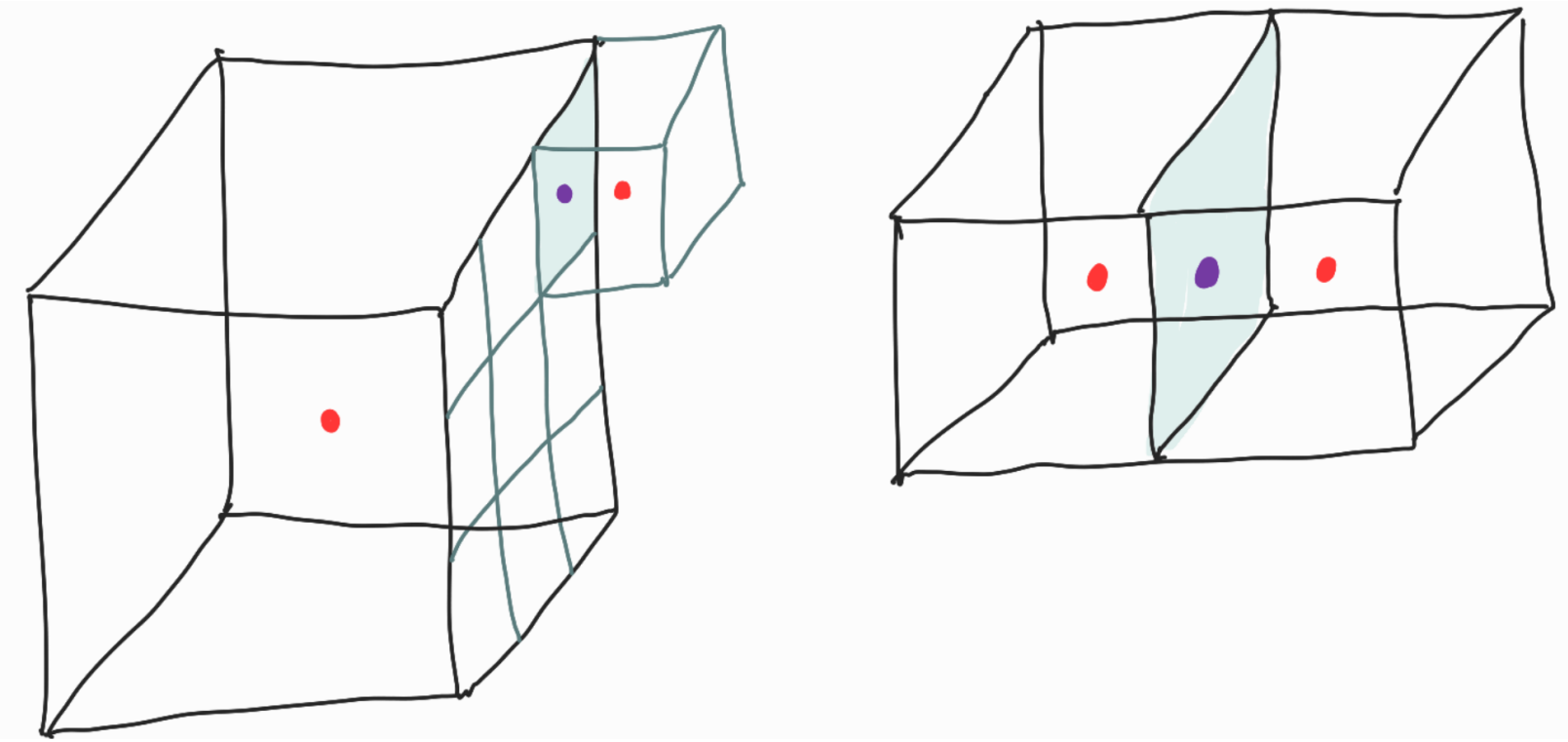
  Why? **Two-point flux approximation**



5. Simulation

- **Jumps** on the simulation on **LGR boundaries**

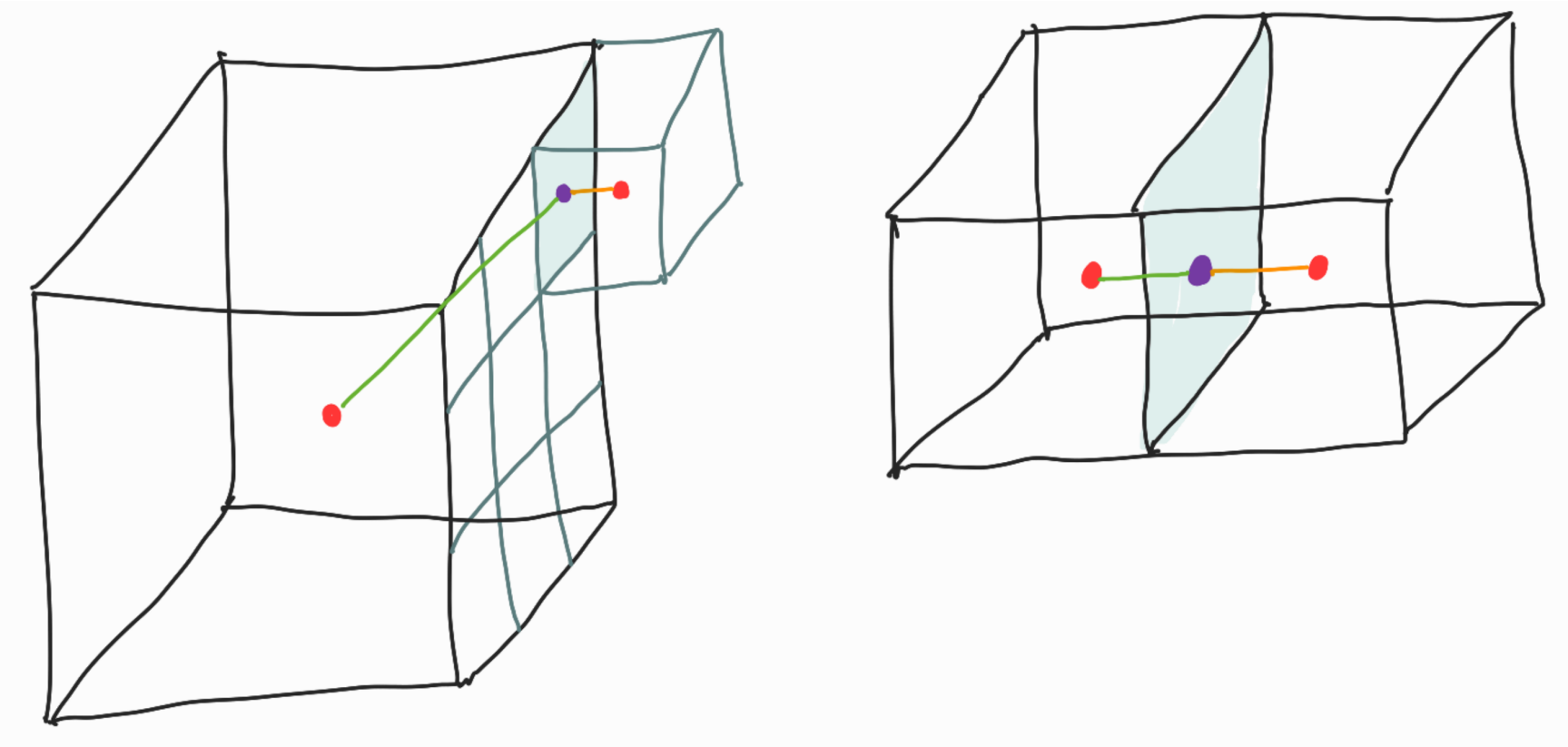
  Why? **Two-point flux approximation**



5. Simulation

- **Jumps** on the simulation on **LRG boundaries**

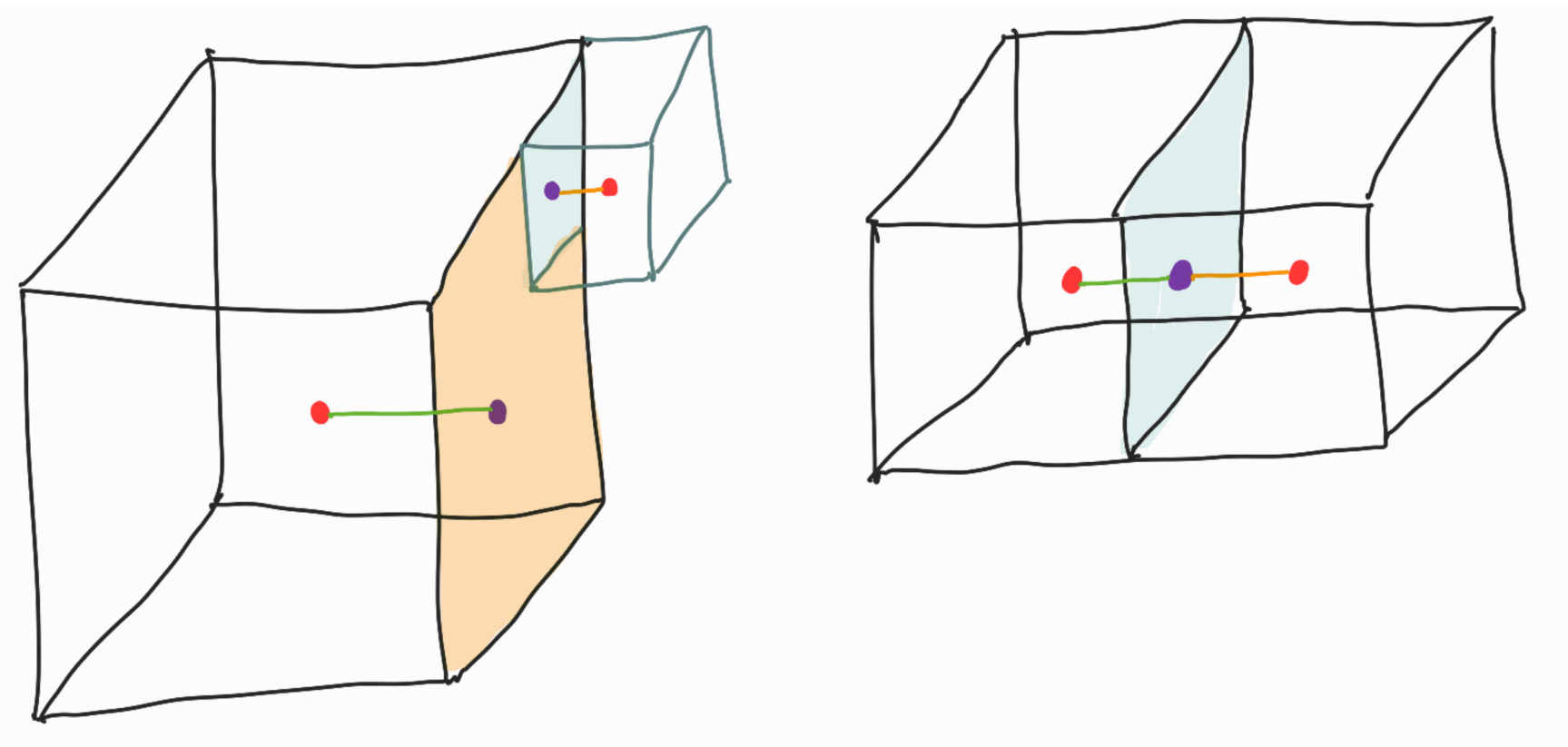
  Why? **Two-point flux approximation**



5. Simulation

- **Jumps** on the simulation on **LGR boundaries**

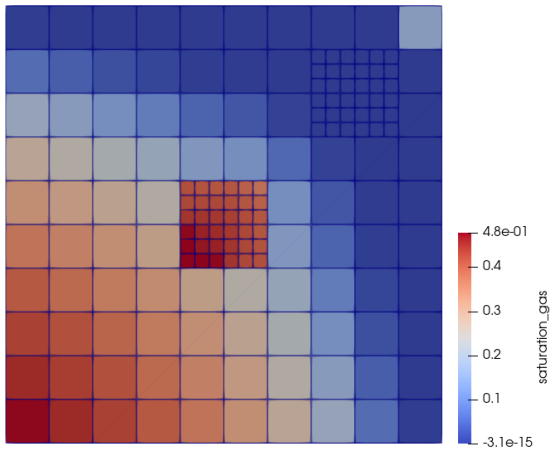
  Solution **Track parent intersection**

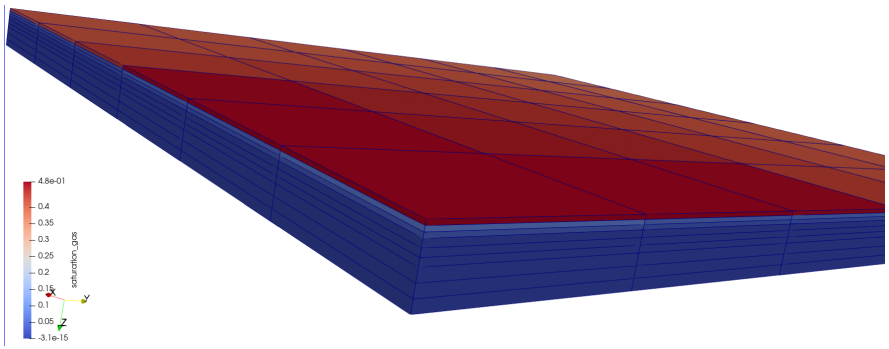


5. Simulation

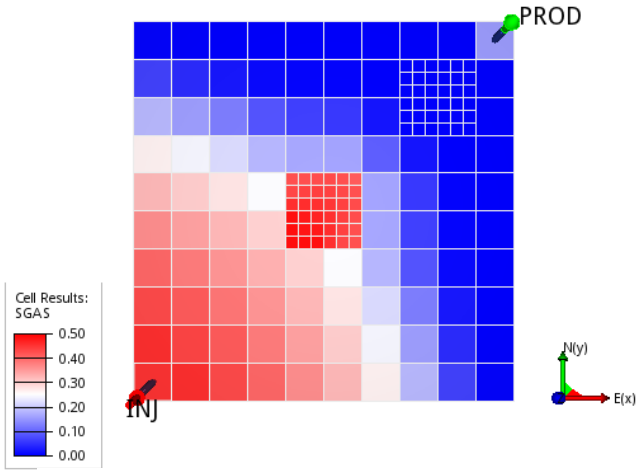
- **Jumps** on the simulation for **saturation gas/oil**

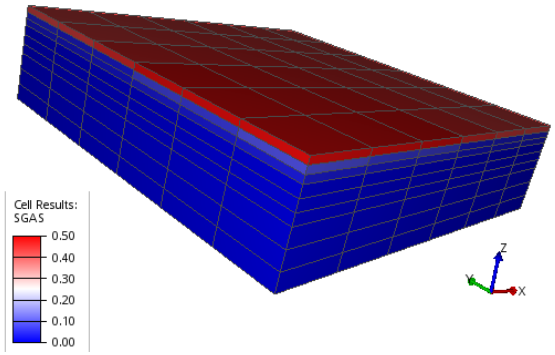
OPM Flow





OTHER

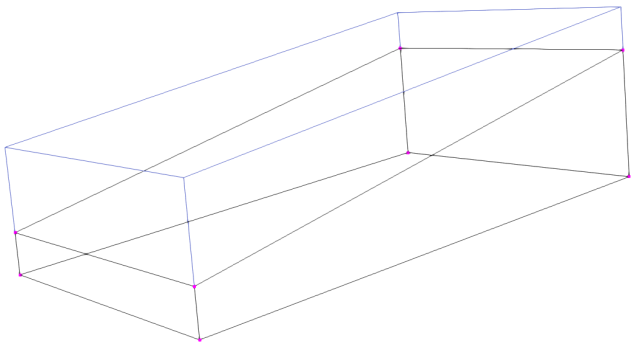




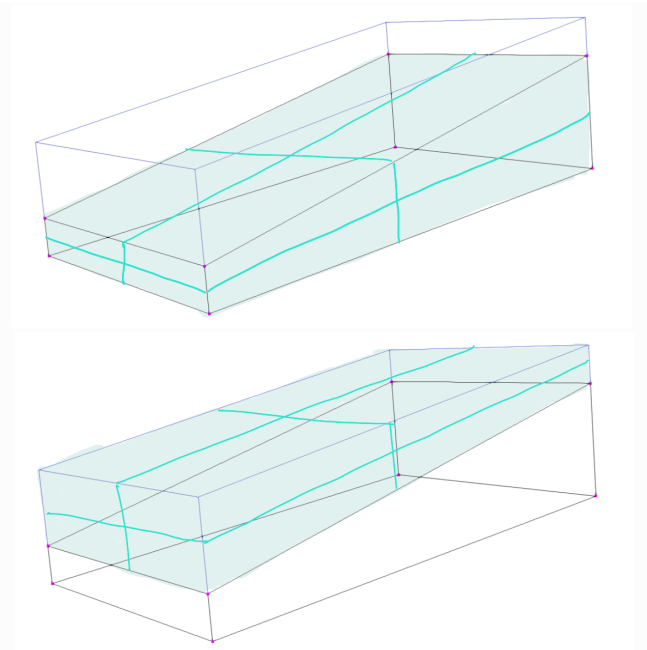
How far from the goal?

How far from the goal? - **Irregular** CpGrid

Two-deformed-cells

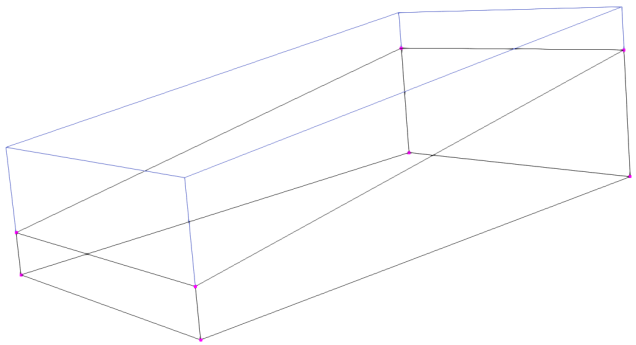


Expectation



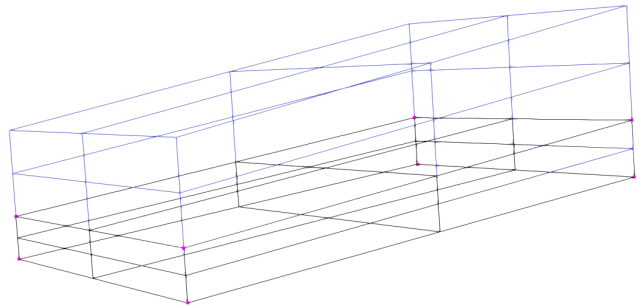
How far from the goal? - **Irregular** CpGrid

Two-deformed-cells



	Point ID	Points			Points_Magnitude
0	0	0	0	0	0
1	1	6	0	0.5	6.0208
2	2	0	4	0	4
3	3	6	4	0.5	7.22842
4	4	0	0	0.5	0.5
5	5	6	0	2	6.32456
6	6	0	4	0.5	4.03113
7	7	6	4	2	7.48331

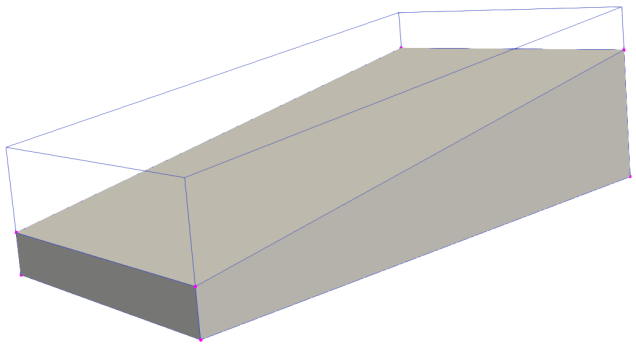
LGR with OPM Flow



	Point ID	Points			Points_Magnitude
0	0	0	0	0	0
1	8	6	0	0.5	6.0208
2	12	0	4	0	4
3	16	6	4	0.5	7.22842
4	18	0	0	0.5	0.5
5	22	6	0	1.16667	6.11237
6	24	0	4	0.5	4.03113
7	26	6	4	1.16667	7.30487

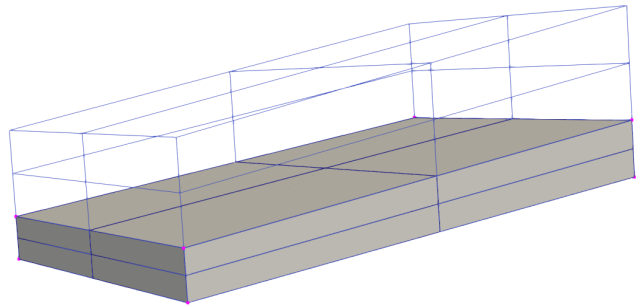
How far from the goal? - **Irregular** CpGrid

Two-deformed-cells



	Point ID	Points			Points_Magnitude
0	0	0	0	0	0
1	1	6	0	0.5	6.0208
2	2	0	4	0	4
3	3	6	4	0.5	7.22842
4	4	0	0	0.5	0.5
5	5	6	0	2	6.32456
6	6	0	4	0.5	4.03113
7	7	6	4	2	7.48331

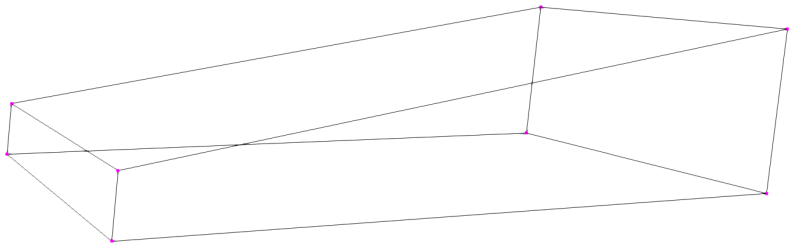
LGR with OPM Flow



	Point ID	Points			Points_Magnitude
0	0	0	0	0	0
1	8	6	0	0.5	6.0208
2	12	0	4	0	4
3	16	6	4	0.5	7.22842
4	18	0	0	0.5	0.5
5	22	6	0	1.16667	6.11237
6	24	0	4	0.5	4.03113
7	26	6	4	1.16667	7.30487

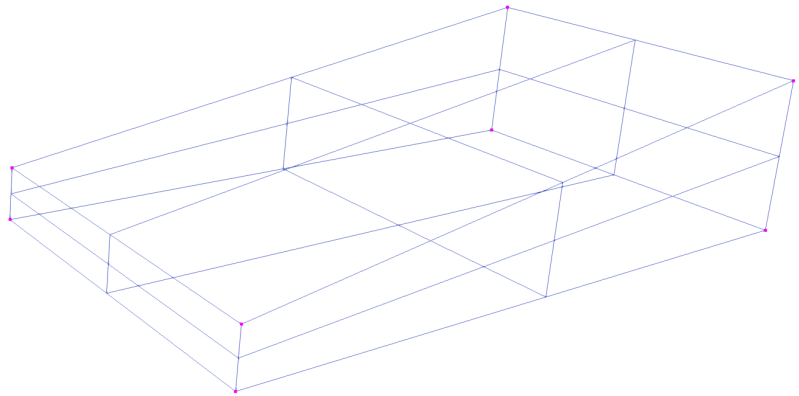
How far from the goal? - **Irregular** CpGrid

Single-deformed-cell



	Point ID	Coordinates			Coordinates_Magnitude
0	0	0	0	0	0
1	1	6	0	0.5	6.0208
2	2	0	4	0	4
3	3	6	4	0.5	7.22842
4	4	0	0	0.5	0.5
5	5	6	0	2	6.32456
6	6	0	4	0.5	4.03113
7	7	6	4	2	7.48331

LGR with OPM Flow



	Point ID	Points			Points_Magnitude
0	0	0	0	0	0
1	8	6	0	0.5	6.0208
2	12	0	4	0	4
3	16	6	4	0.5	7.22842
4	18	0	0	0.5	0.5
5	22	6	0	2	6.32456
6	24	0	4	0.5	4.03113
7	26	6	4	2	7.48331

Single-deformed-cell LGRs ✓

How far from the goal?

Project goals

✓ Support **simulation** for **CpGrid with LGRs**

(Regular) Cartesian CpGrid	**(Irregular) CpGrid**
• active cells • no wells in LGRs • disjoint LGRs	• active cells • no wells in LGRs • disjoint LGRs • each LGR comes from a **single-deformed-cell**

✗ Provide **output files** *still missing!*

What's next...

What's next ...

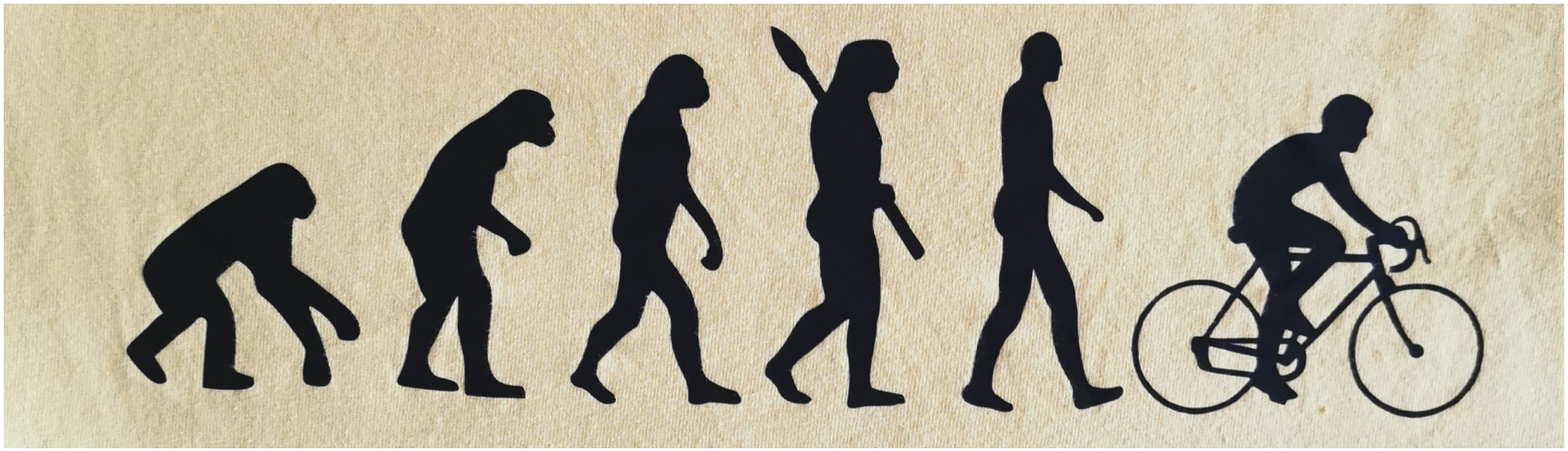
In progress

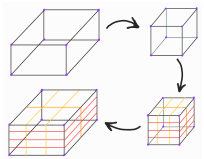
- Understanding/fixing *saturation-gas*-**jumps on the simulation**
- **Irregular CpGrids** for the case of **single deformed cells**

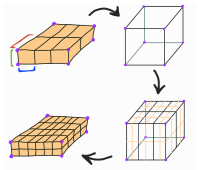
Not started yet

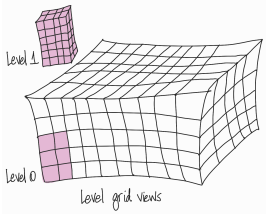
- Provide **output files**
- Allow **inactive** cells
- In **DUNE**'s perspective
  - **mark** subset of leaf grid view elements to refine or coarsen
  - support **adapt(), preAdapt(), postAdapt()**, ...
- LGRs for **general irregular** CpGrids
- Simulation in **parallel**?
- Allow **deformed cells** with **fewer corners** (currently, 8 corners)
- (audience wishes?)...

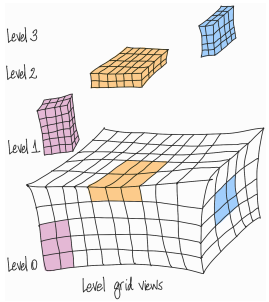
LGR evolution for CpGrids

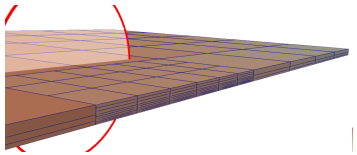




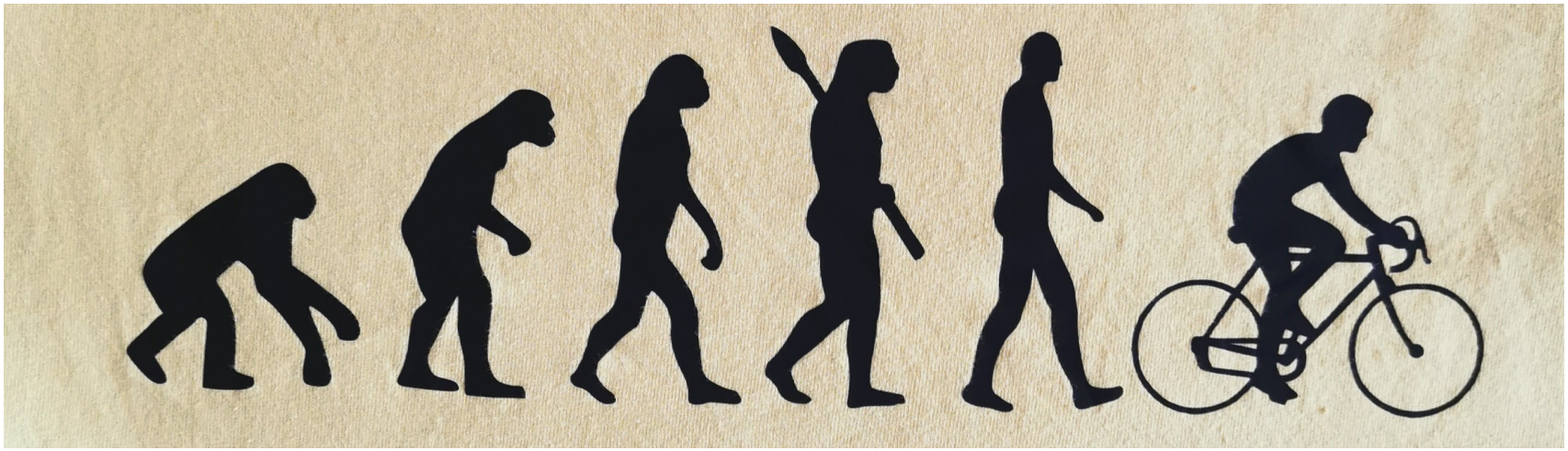


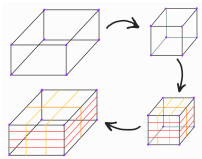


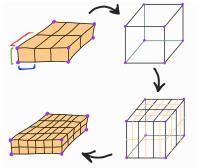


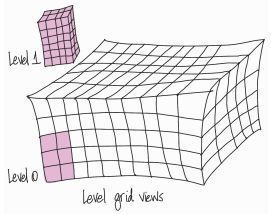


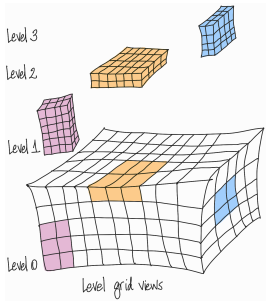
LGR evolution for CpGrids

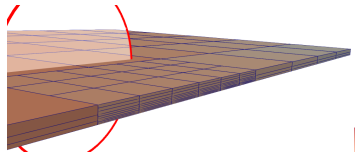












Thank you for your attention!