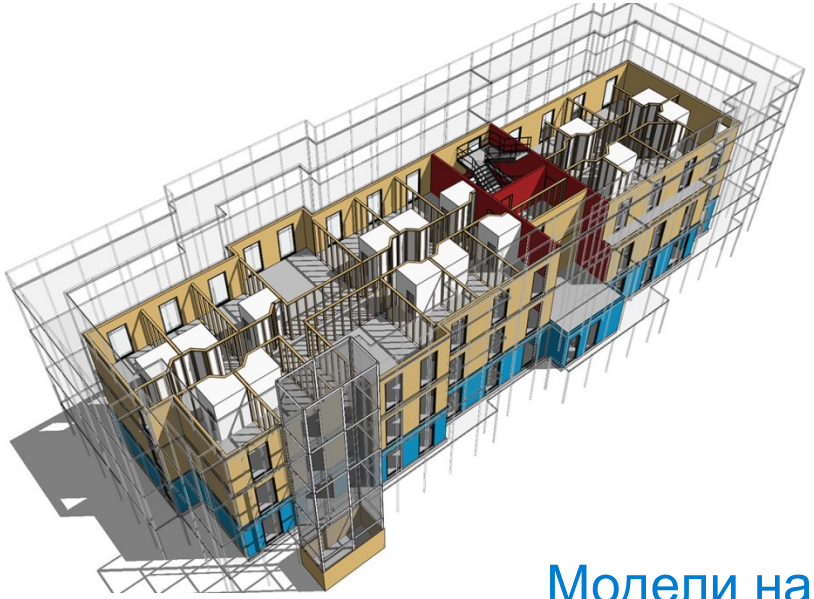
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОСКОВСКИЙ
ГОСУДАРСТВЕННЫЙ
СТРОИТЕЛЬНЫЙ
УНИВЕРСИТЕТ

Искусственный интеллект в информационных моделях, основанных на топологическом подходе

Рожков Александр Николаевич, ст. преподаватель rozhkovalex@hotmail.com
Галишникова Вера Владимировна, профессор, д.т.н.

Традиционный подход в BIM



Информационные модели зданий основаны на отраслевых базовых классов (IFC).
Экземпляр отраслевого базового класса описывает топологию и геометрию оригинала и определяет материальные, функциональные и другие свойства.
Отображая компоненты построенных объектов в экземпляры IFC, программные платформы делают часть информации явной, которая раньше была неявной

Модели на основе отраслевых базовых классов строятся путем поэтапной сборки их компонентов

- может произойти столкновение нового компонента с существующими компонентами модели и непреднамеренные зазоры между компонентами
- Чтобы избежать этой проблемы, необходимо определить его местоположение относительно каждого из существующих компонентов модели, что является объемной задачей
- избегаются неограниченные компоненты модели, что требует специальных теорий и методов на границе
- Физические явления, такие как затухание звука и вибрации, зависят от моделирования неограниченного пространства

Цель исследовательского проекта –

разработка общей теории, которая позволяет полностью описывать топологию и геометрию очень сложных многогранных объектов, используя только явную информацию, содержащуюся в моделях.

Цель данной работы –

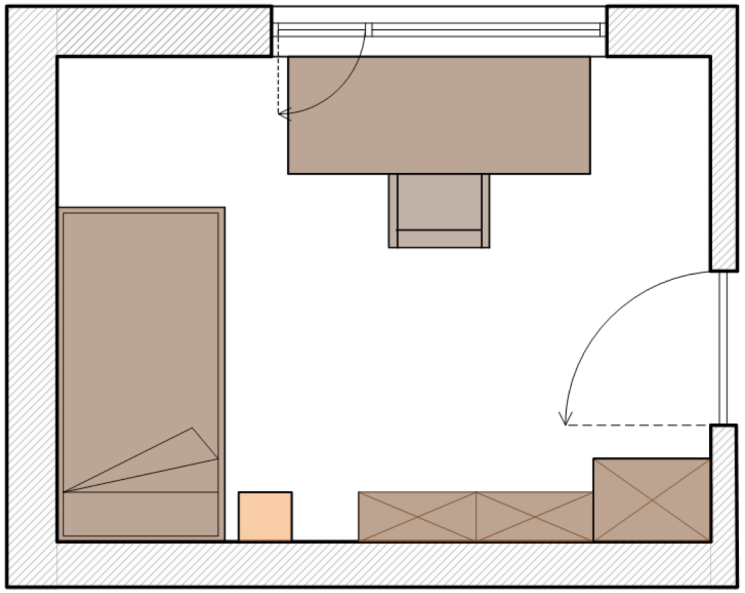
сделать геометрическую информацию цифровой модели оригинала максимально явной, надежной и полной.

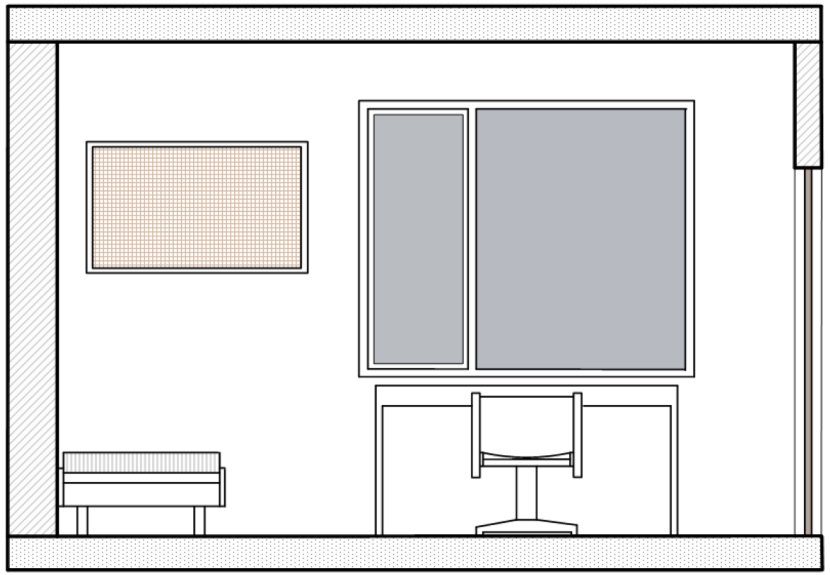
Задачи:

- **Разработка эффективного метода построения топологических таблиц линейного комплекса.**
- **Разработка структур данных, удобных для описания комплексов и построения топологических таблиц.**
- **Оценка сложности разработанного алгоритма.**
- **Оптимизация алгоритма с применением параллельного программирования на GPU**

Явная и неявная информация

План и разрез комнаты с мебелью



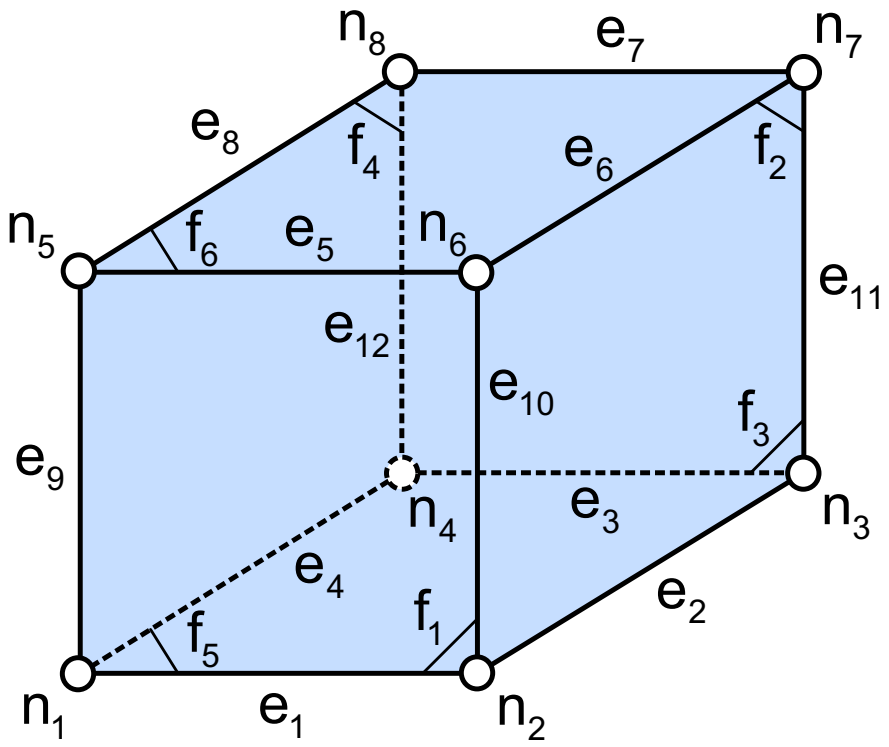


Явная информация – это информация, которая описывается переменными.

Неявная информация получается косвенным путем, например, при помощи алгоритмов, или при интерпретации чертежа.

Неявная информация – потенциальный источник ошибок.

Топологический подход



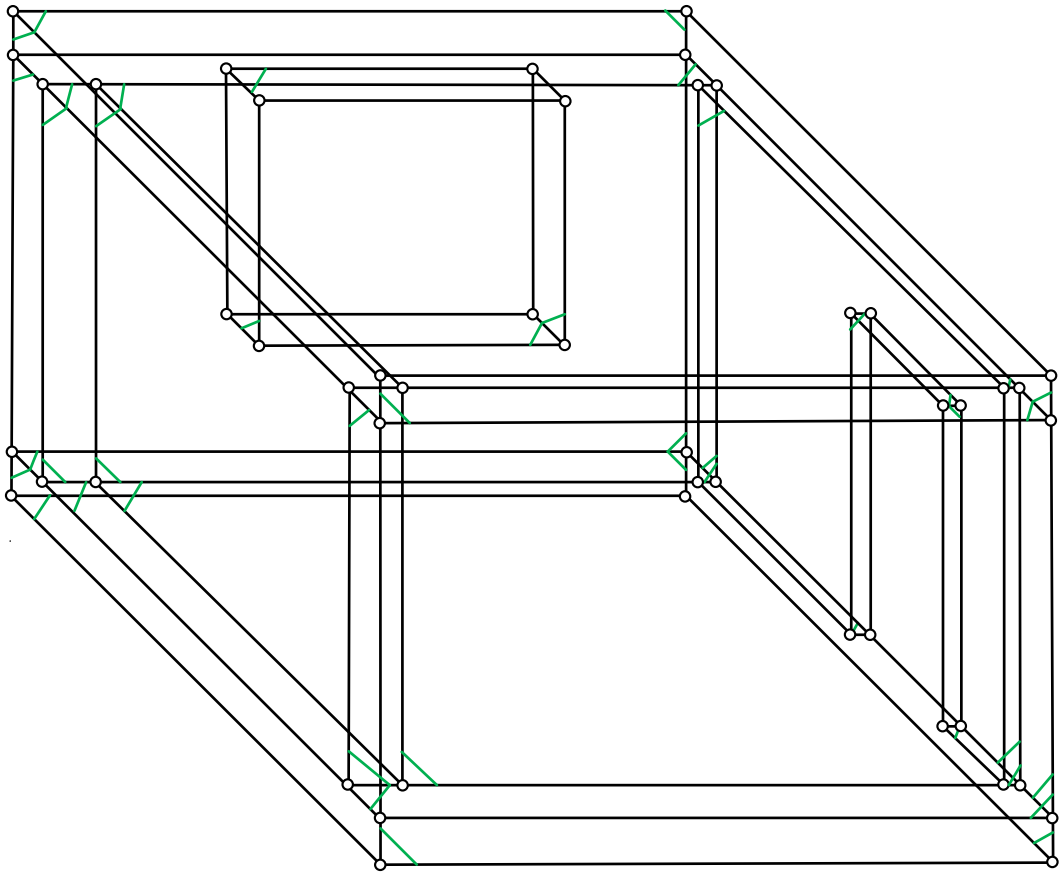
Узел (точка) – элемент 1-го ранга.

Ребро (отрезок прямой линии) – элемент 2-го ранга.

Грань (плоская область, ограниченная по меньшей мере одной замкнутой многоугольной кривой) – элемент 3-го ранга.

Ячейка (объем, ограниченный по меньшей мере одной замкнутой многогранной поверхностью, состоящей из граней) – элемент 4-го ранга

Линейный комплекс



Линейный комплекс для комнаты с мебелью

Линейный комплекс – это 3D объект

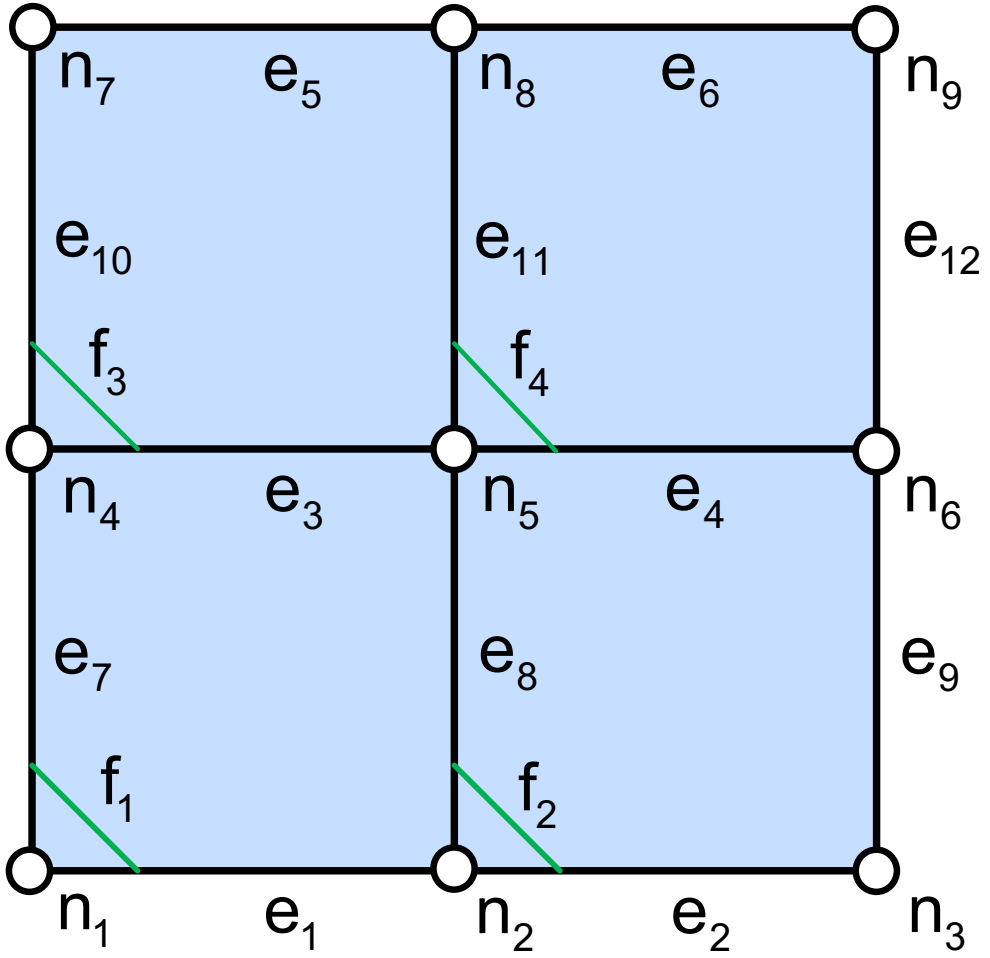
Топология линейного комплекса явно предоставляет все данные.

Каждый элемент комплекса определен уникальным именем.

Топологические таблицы являются инструментом для анализа и описания топологии линейных комплексов.

Топологический контакт

- Все элементы линейного комплекса являются геометрическим многообразиями
- Спецификация комплексов систематизируются путем присвоения рангов типам элементов
- Два типа контакта элемента A с рангом k и элемента B с рангом m определяются следующим образом:
- 1) Ранг k>m: Элемент B находится в контакте с элементом A, если набор точек границы элемента A содержит набор точек элемента B.
- 2) Ранг k<m: Элемент A находится в контакте с элементом B, если набор точек границы элемента B содержит набор точек элемента A.



Топологический контакт между узлами, ребрами и гранями

Топологический таблицы

T_{km}		rank m , domain type			
		0 node	1 edge	2 face	3 cell
rank k, domain type	0 node	T_{00}	T_{01}	T_{02}	T_{03}
	1 edge	T_{10}	T_{11}	T_{12}	T_{13}
	2 face	T_{20}	T_{21}	T_{22}	T_{23}
	3 cell	T_{30}	T_{31}	T_{32}	T_{33}

Матрица топологических таблиц T_{km} для линейных комплексов

Топологические таблицы описывают топологию линейного комплекса.

Есть три базовые таблицы:

- **Ребро-узел**
- **Грань-ребро**
- **Ячейка-грань**

Структуры данных

- Класс для элемента ранга n определяется только использованием элементов n-1 ранга. Элементы n ранга (кроме узлов) содержат в себе как атрибуты ссылки на элементы n-1 ранга.

- Идентификаторы имен и ссылки на объекты хранятся в словарях, для каждого класса свой словарь.

- Топологические таблицы хранятся в отсортированном словаре.

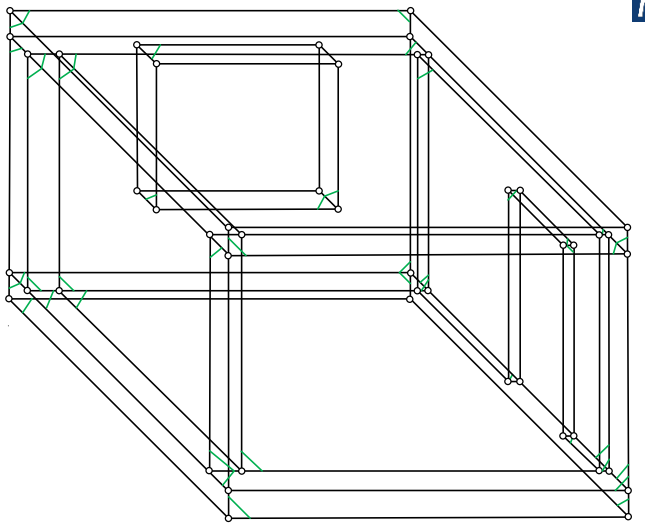
Алгоритм построения топологических таблиц

- **Разработаны команды для ввода элементов в комплекс**

- **Все таблицы строятся внутри одного метода.**
- **Выбирается структура максимального ранга (cellmap), такой прием позволяет сократить время построения всех таблиц**
- **Поскольку любой объект класса Edge, Face или Cell ссылается только на объекты следующего более низкого ранга, 12 таблиц могут быть построены в четырех вложенных циклах:**
  1) **Цикл по ячейкам**
  2) **Цикл по граням ячейки *c***
  3) **Цикл по ребрам *e* грани *f* ячейки *c***
  4) **Цикл по узлам *n* ребра *e* грани *f* ячейки *c***

Сложность и эффективность алгоритма

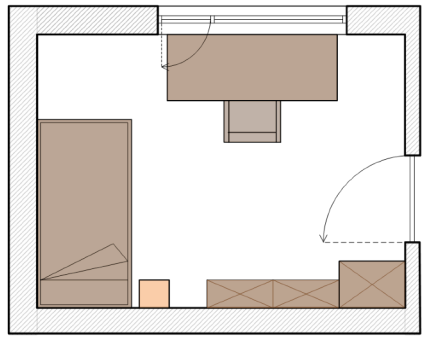
В ходе построения таблиц для примера с жилой комнатой количество операции записи в топологические таблицы равно 2655 при учете того, что весь процесс построения был выполнен в 337 шагов и модель состоит из 200 элементов.

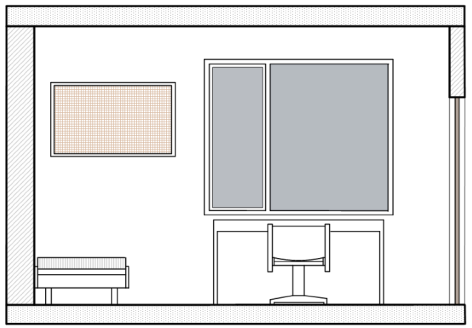


Надежность достигается за счет того, что 1 топологическое свойство модели изменяется в течении 1 рабочего шага.

$$N_{total} = 2N_cN_f + 4N_cN_fN_e + 12N_cN_fN_e = 2N_cN_f(1 + 8N_e)$$
$$\approx 16N_cN_fN_e \approx O(N_c\ N_fN_e) \approx 2400$$

Среднее количество граней на ячейку N_f и среднее количество ребер на грань N_e не зависят от размера комплекса. Таким образом окончательно сложность полученного алгоритма $O(N_c)$.





Искусственный интеллект с точки зрения аппаратного обеспечения

- Для решения задач с достаточной степенью точности требуются значительные вычислительные ресурсы (многопроцессорные системы), которые являются дорогостоящими, энергозатратными установками с ограниченным доступом.
- Новым подходом в решении задачи является применение графических ускорителей (GPU), состоящих из нескольких тысяч процессоров, разработанных специально для параллельных вычислений.
- Применение GPU позволяет сократить время вычислений и получать результаты со степенью точности, сопоставимой с суперкомпьютерами.

Суперкомпьютер Spiking Neural Network Architecture

Графический ускоритель RTX 3090 Founders Edition

PyCuda оптимизация

- **Реализация PyCUDA ядра: наиболее затратная операция построения топологических таблиц перенесена для вычисления на GPU**

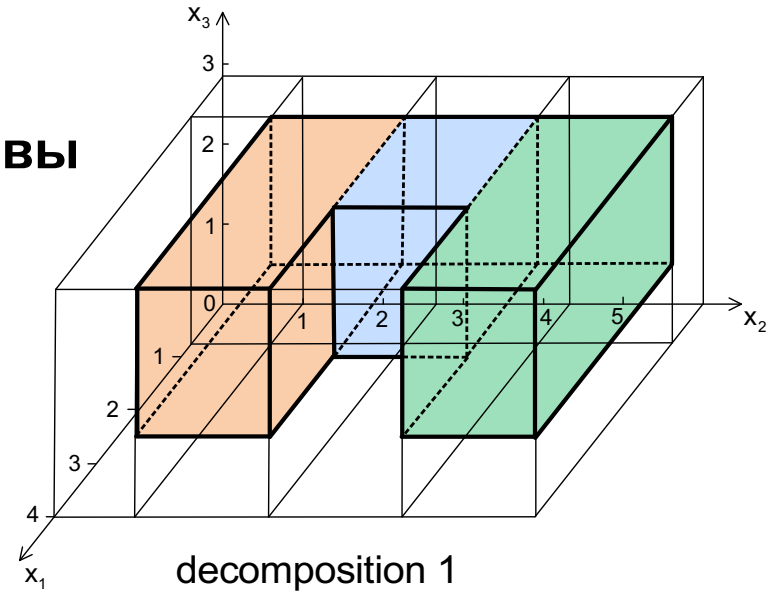
```python
import numpy as np
import matplotlib.pyplot as plt
import pycuda.autoinit
import pycuda.driver as drv
import pycuda.gpuarray as gpuarray

from pycuda.compiler import SourceModule
```

```python
# CUDA ядро
mod = SourceModule("""
__global__ void generate_tables_kernel(int *cell_faces, int *face_edges, int *node_cells,
                                       int *node_faces, int *node_edges, int *edge_nodes,
                                       int *edge_faces, int *edge_cells, int *face_nodes,
                                       int *cell_nodes, int *cell_edges,
                                       int num_cells, int num_faces, int num_edges, int num_nodes) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;

    if (idx < num_cells) {
        // Обход ячеек
        // Обход граней
        // Обход ребер
        // Обход ячеек
    }
}
""")

generate_tables_kernel = mod.get_function("generate_tables_kernel")
```

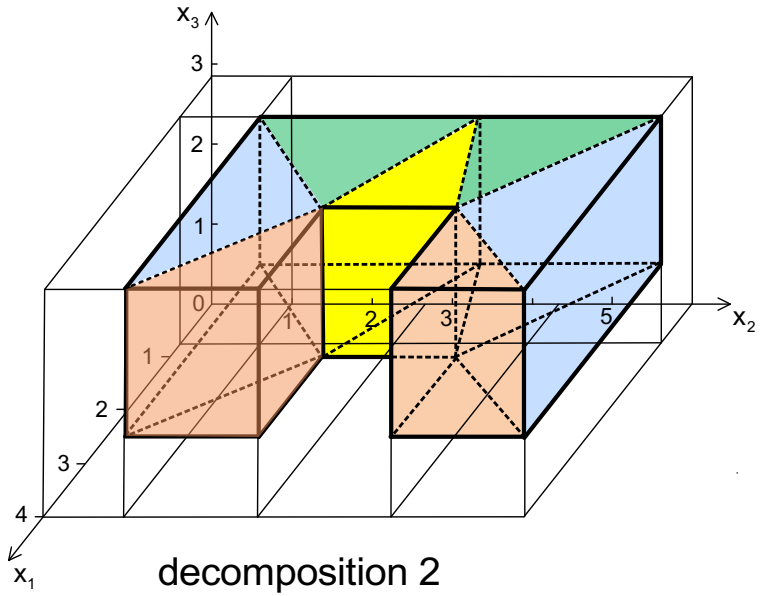
Time_cpu	≈ 120 секунд
Time_gpu	≈ 10 секунд

ИИ на основе топологического подхода

- **Декомпозиция на примитивы**



decomposition 1

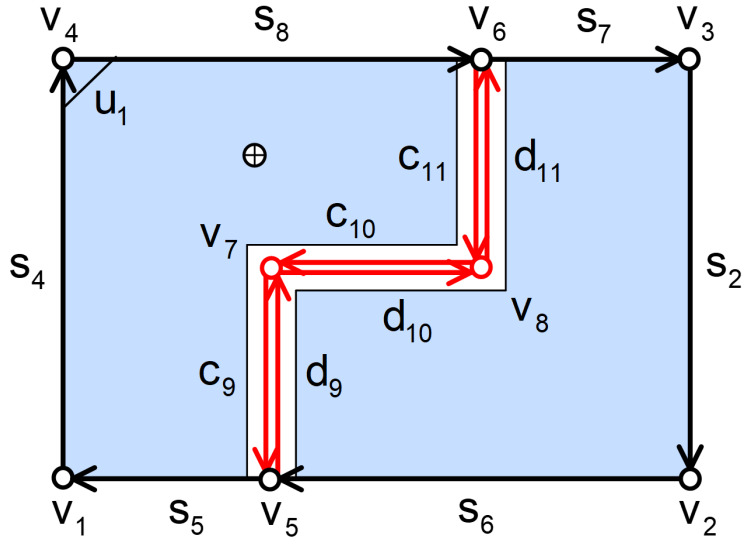


decomposition 2

- **Навигация в линейных комплексах**

- **Выявление структур, подверженных влиянию соседних единиц, которые вышли из строя.**

- **Поддержание полной/частичной работоспособности всей системы в аварийных ситуациях**



Выводы:

Топологические таблицы могут быть эффективно построены для линейных комплексов, представляющих целые здания.

Сложность алгоритма построения таблицы, разработанного в проекте, линейна по количеству ячеек и, следовательно, очень эффективна.

Топологические таблицы не делают все свойства оригинала явными в модели. Замкнутые полигональные кривые границ граней и замкнутые многогранные поверхности границ ячеек явно не указаны в таблицах.

Произведена оптимизация с использованием архитектуры CUDA

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

СПАСИБО ЗА ВНИМАНИЕ!

Рожков Александр
rozhkovalex@hotmail.com

Метод для построения топологических таблиц

```
public static void generateTables(Map<String, Cell> map) {
    for (Map.Entry<String, Cell> pair : map.entrySet()) {
        //filling next tables
        topCellFace.put(cell.name, cell.faces);
        for (Face face : cell.faces) {
                //filling next tables
                topFaceEdge.put(face.name, face.edges);
                topFaceCell.put(face.name, set);
                for (Edge edge : face.edges) {
                        //filling next tables
                        topNodeCell.put(edge.node1(2).name, set);
                        topNodeFace.put(edge.node1(2).name, set);
                        topNodeEdge.put(edge.node1(2).name, set);
                        topEdgeNode.put(edge.name, edgenodeset);
                        topEdgeFace.put(edge.name, set);
                        topEdgeCell.put(edge.name, set);}}

                topFaceNode.put(face.name, facenodeset);}
        topCellNode.put(cell.name, cellnodeset);
        topCellEdge.put(cell.name, celledgeset);}}
```

1. Loop over the cells of the element group:
- perform the following loop for each cell c

2. Loop over the faces f of cell c:
- add face f to the cell-face-table with key c
- add cell c to the face-cell-table with key f
- perform the following loop for each face f

3. Loop over the edges e of face f of cell c:
- add edge e to the cell-edge-table with key c
- add edge e to the face-edge-table with key f
- add face f to the edge-face-table with key e
- add cell c to the edge-cell-table with key e
- perform the following loop for each edge e

4. Loop over the nodes n of edge e of face f of cell c:
- add node n to the cell-node-table with key c
- add node n to the face-node-table with key f
- add node n to the edge-node-table with key e
- add cell c to the node-cell-table with key n
- add face f to the node-face-table with key n
- add edge e to the node-edge-table with key n

PyCuda

```python
import numpy as np
import matplotlib.pyplot as plt
import pycuda.autoinit
import pycuda.driver as drv
import pycuda.gpuarray as gpuarray

from pycuda.compiler import SourceModule
```

```python
# CUDA ядро
mod = SourceModule("""
__global__ void generate_tables_kernel(int *cell_faces, int *face_edges, int *node_cells,
                                       int *node_faces, int *node_edges, int *edge_nodes,
                                       int *edge_faces, int *edge_cells, int *face_nodes,
                                       int *cell_nodes, int *cell_edges,
                                       int num_cells, int num_faces, int num_edges, int num_nodes) {
    int cell_index = threadIdx.x + blockIdx.x * blockDim.x;

    if (cell_index < num_cells) {
            Cell* cell = cells[cell_index];
            std::unordered_set<Node*> cellnodeset;
            std::unordered_set<Edge*> celledgeset;

            // Cell-Face relationships
            lc->topCellFace[cell] = std::unordered_set<Face*>(cell->faces.begin(), cell->faces.end());

            // Loop through every face of the cell
            for (Face* face : cell->faces) {
                std::unordered_set<Node*> facenodeset;

                // Loop through every edge of the face
                for (Edge* edge : face->edges) {
                    // Node-Cell relationships
                    lc->add_to_map_set(lc->topNodeCell, edge->node1, cell);
                    lc->add_to_map_set(lc->topNodeCell, edge->node2, cell);

                    // Cell-Node relationships
                    cellnodeset.insert(edge->node1);
                    cellnodeset.insert(edge->node2);

                    // Node-Face relationships
                    lc->add_to_map_set(lc->topNodeFace, edge->node1, face);
                    lc->add_to_map_set(lc->topNodeFace, edge->node2, face);

                    // Face-Node relationships
                    facenodeset.insert(edge->node1);
                    facenodeset.insert(edge->node2);

                    // Node-Edge relationships
                    lc->add_to_map_set(lc->topNodeEdge, edge->node1, edge);
                    lc->add_to_map_set(lc->topNodeEdge, edge->node2, edge);

                    // Edge-Node relationships
                    lc->topEdgeNode[edge] = std::unordered_set<Node*>{edge->node1, edge->node2};

                    // Edge-Face relationships
                    lc->add_to_map_set(lc->topEdgeFace, edge, face);

                    // Edge-Cell relationships
                    lc->add_to_map_set(lc->topEdgeCell, edge, cell);
                }

                // Face-Node relationships
                lc->topFaceNode[face] = facenodeset;

                // Face-Edge relationships
                lc->topFaceEdge[face] = std::unordered_set<Edge*>(face->edges.begin(), face->edges.end());

                // Face-Cell relationships
                lc->add_to_map_set(lc->topFaceCell, face, cell);
            }

            // Cell-Node relationships
            lc->topCellNode[cell] = cellnodeset;

            // Cell-Edge relationships
            lc->topCellEdge[cell] = std::unordered_set<Edge*>(celledgeset.begin(), celledgeset.end());
        }
}
""")

generate_tables_kernel = mod.get_function("generate_tables_kernel")
```

Построенная топологическая таблица

Node-Edge table					
n1	e1, e4, e26	n17	e21, e22, e25, e52	n33	e43, e63, e64, e68
n2	e1, e2, e31	n18	e20, e23, e49, e50	n34	e50, e53, e67, e70
n3	e2, e3, e40	n19	e22, e23, e24, e51	n35	e51, e69, e70, e84
n4	e3, e4, e49	n20	e6, e8, e25, e29	n36	e52, e68, e69, e85
n5	e5, e6, e24, e28	n21	e30, e55, e83	n37	e36, e71, e72
n6	e5, e7, e26, e27	n22	e35, e55, e61	n38	e37, e71, e73
n7	e7, e9, e31, e32	n23	e44, e61, e66	n39	e38, e72, e74
n8	e9, e10, e11, e33	n24	e53, e66, e83	n40	e39, e73, e74
n9	e8, e10, e12, e34	n25	e27, e30, e54, e58	n41	e45, e75, e77
n10	e11, e13, e14, e36	n26	e28, e54, e56, e84	n42	e46, e75, e76
n11	e12, e13, e37	n27	e29, e56, e57, e85	n43	e45, e79, e81
n12	e14, e15, e16, e38	n28	e32, e35, e58, e60	n44	e46, e79, e80
n13	e15, e17, e39	n29	e33, e59, e60, e62	n45	e47, e81, e82
n14	e16, e18, e19, e42	n30	e34, e57, e59, e63	n46	e48, e80, e82
n15	e17, e18, e21, e43	n31	e41, e44, e65, e67	n47	e47, e77, e78
n16	e19, e20, e40, e41	n32	e42, e62, e64, e65	n48	e48, e76, e78

ЦИФРА СТРОЙ

XV Академические чтения, посвященные памяти академика РААСН Осипова Г.Л.

Научно-практическая конференция «Перспективы использования искусственного интеллекта в градостроительной деятельности», Москва, 2 – 3 июля 2024 г.

Модераторы:

Валерия Мозганова, Радиостанция Business FM, руководитель отдела «Недвижимость»

Евгений Карант, НИИСФ РААСН, ведущий инженер

Полный список докладов доступен на сайте ЦифраСтрой по ссылке

https://cifrastroy.ru/news/buduschee-iskusstvennogo-intellekta-v-gradostroitelstve