2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

doi 10.26089/NumMet.v27r212 УДК 004.428.4

Опыт использования библиотеки p4est для генерации октосетки

А. В. Соловьев
Московский государственный университет имени М. В. Ломоносова,
факультет вычислительной математики и кибернетики,
Москва, Российская Федерация
ORCID: 0000-0002-9781-9527, e-mail: solovjev@cs.msu.ru

Аннотация: На примере задачи управляемого построения октосетки в области сложной формы описаны функции библиотеки с открытым исходным кодом `p4est`, предназначенной для распределенной работы с окто- и квадросетками. Приведены сигнатуры основных функций, показаны примеры их возможного использования, описаны нюансы взаимодействия. Рассмотрены возможности библиотеки по построению базовой сетки, управляемому измельчению и огрублению сетки, поддержанию принципа смежности 2 : 1, выравниванию нагрузки на процессоры и другие вопросы.

Ключевые слова: октосетка, квадросетка, октодерево, квадродерево, построение октосетки, библиотека p4est.

Для цитирования: Соловьев А.В. Опыт использования библиотеки p4est для генерации октосетки // Вычислительные методы и программирование. 2026. **27**, № 2. 172–193. doi 10.26089/NumMet.v27r212.

Experience using the p4est library for octogrid generation

Andrey V. Solovjev
Lomonosov Moscow State University,
Facility of Computational Mathematics and Cybernetics,
Moscow, Russia
ORCID: 0000-0002-9781-9527, e-mail: solovjev@cs.msu.ru

Abstract: Using the example of controlled octogrid construction in a complex domain, this article describes the functions of the open-source p4est library, designed for distributed work with octogrids and quadgrids. The main function signatures are provided, examples of their possible use are shown, and the nuances of their interaction are described. The library's capabilities for base grid construction, controlled mesh refinement and coarsening, maintaining the 2 : 1 adjacency principle, processor load balancing, and other issues are discussed.

Keywords: octogrid, quadgrid, octree, quadtree, octogrid construction, p4est library.

For citation: A. V. Solovjev, "Experience using the p4est library for octogrid generation," Numerical Methods and Programming. **27** (2), 172–193 (2026). doi 10.26089/NumMet.v27r212.

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

1. Введение. При численном решении систем уравнений в частных производных заметное место занимают задачи построения расчетных сеток в 2D и 3D рабочих областях, часто имеющих сложную конфигурацию. Используются различные сетки — регулярные и нерегулярные, состоящие из объектов разной формы, подвижные и неподвижные, фиксированные и адаптивные и пр. Значительное место среди расчетных сеток заняли сетки, базирующиеся на понятии окто- (в 3D) или квадро- (в 2D) деревьях (см., например, [1]). Такие сетки получили название, соответственно, октосетки и квадросетки.

Определение окто- и квадросеток обычно дается через способ их построения. В самом общем случае выбирается некоторая начальная сетка (называемая базовой), состоящая из четырехугольников (в 2D) или из гексаэдров (в 3D), каждая из ячеек которой может быть разделена в двумерном случае средними линиями на 4 меньших ячейки, а в трехмерном — средними поверхностями на 8. Такое деление может быть повторным. На практике чаще всего в качестве базовой сетки выбирается либо параллелепипед (прямоугольник), либо, реже, фигура, полученная из него неким гладким преобразованием.

Сеточные методы, использующие октосетки (все сказанное имеет равное отношение и к квадросеткам), имеют как положительные, так и отрицательные особенности. К положительным можно отнести наилучшую с точки зрения аппроксимации форму ячейки — параллелепипед или прямоугольник, возможность проводить локальную адаптацию сетки, не затрагивающую остальную расчетную область, выигрыш в эффективности кода при замене поверхностного (контурного) интеграла на конечные разности в конечно-объемных методах и пр. Недостатком является невозможность отслеживания октосеткой сложных границ расчетной области. В таких случаях приходится либо иметь часть ячеек другой формы, либо учитывать “зубчатость” границы. Применяются также специальные методы совмещения точной границы расчетной области и приграничных ячеек октосетки, такие как метод внедренных границ (immersed boundaries) [2], метод сдвинутых границ (shifted boundaries) [3] и др.

Несмотря на наличие готовых средств построения сеток различных видов в имеющихся пакетах программ, часто разработчикам новых кодов целесообразно иметь собственные средства построения сеток. Отмеченные выше успехи в преодолении негативных свойств методов, использующих октосетки, делают все более актуальной разработку кодов как для эффективного распараллеленного построения статических октосеток с управляемым сгущением к границам и областям с особенностями в решении, так и динамической адаптации сетки. Разработаны и реализованы различные алгоритмы построения и адаптации окто- и квадросеток [4, 5].

Среди доступных для использования открытых кодов можно отметить библиотеку `p4est` (читается pi-forest) [6–9], позволяющую выполнять все необходимые операции по начальному построению и динамической адаптации как окто-, так и квадросеток. Библиотека способна работать в однопроцессорном и многопроцессорном режимах с использованием стандартной MPI-параллельности (в [6] говорится об опыте запуска на 220320 процессорах для построения сетки из 5.13×10^{11} ячеек). В библиотеку `p4est` входят функция параллельного чтения из файла сетки вместе с ассоциированными данными пользователя (число читающих процессоров не зависит от числа пишущих), функция выравнивания нагрузки на процессоры, функция измельчения смежных ячеек в соотношении 2 : 1 и другие функции.

Библиотека может представлять интерес для использования при создании CFD-кодов, основанных на окто- и квадросетках. До сих пор в русскоязычной литературе не было публикаций, описывающих использование этой библиотеки. В англоязычной литературе число публикаций также невелико.

В настоящей статье описывается опыт использования библиотеки `p4est` для управляемой генерации начальных сеток. Акцент делается не на практику построения, а именно на использованные механизмы библиотеки — сигнатуры функций, примеры кода, обнаруженные недокументированные особенности. Что касается применения библиотеки для динамической адаптации сетки на этапе решения систем уравнений с частными производными, данные аспекты не вошли в настоящую работу. Автор планирует посвятить этому вопросу отдельную публикацию.

2. Основные сведения о библиотеке p4est. Библиотека `p4est` — это библиотека с открытым исходным кодом, доступная по лицензии GNU GPL версии 2 [10] (ссылку на источник для свободного скачивания библиотеки см. в [11]). Основное назначение библиотеки — MPI-параллельная работа с лесом квадро- или октодеревьев с поддержкой операций адаптационного уточнения сетки (AMR, Adaptive Mesh Refinement). Библиотека предоставляет возможность параллельного хранения сетки вместе с данными, ассоциированными с ячейками, и возможность выполнения некоторых операций над сеткой и ассоциированными данными.

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Обязательным пререквизитом библиотеки `p4est` является библиотека `sc` тех же авторов, что и `p4est`. Исходные коды библиотеки `sc` распространяются в едином с `p4est` дистрибутивном пакете [11] под лицензией LGPL 2.1 [12]. Процедура сборки `sc` входит в сборку библиотеки `p4est`.

Библиотека `p4est` может работать как с квадросеткой в двумерном координатном пространстве, так и с октосеткой в трехмерном, поэтому набор функций библиотеки дублирован. Имена функций и структур данных могут иметь префикс `p4` или `p8` в зависимости от используемой размерности пространства. Исторически развитие библиотеки начиналось с двумерных объектов и с префикса `p4`. Затем была добавлена часть, аналогичная двумерной, но работающая с трехмерными объектами. Те функции и типы, которые вынужденно были переписаны при переходе с 2D на 3D, получили префикс 3D. Но типы данных и функции, которые оказались общими для обеих размерностей, не дублированы и остались с префиксом `p4`, поэтому в коде часто соседствуют префиксы `p4` и `p8`.

Авторы библиотеки предусмотрели возможность использования одного и того же кода как в 2D, так и в 3D. Для обеспечения универсальности служит препроцессорное макроопределение `P4_TO_P8`. Например, конструкция `P4_TO_P8(p4est_t)` может означать либо `p4est_t`, либо `p8est_t` в зависимости от наличия макропеременной P4EST.

Примеры программ, поставляемые вместе с библиотекой, демонстрируют, как одна и та же программа функционирует как для двумерного, так и для трехмерного случая в зависимости от этой препроцессорной переменной. Переход между 2D и 3D не является динамическим (если в коде пользователя нет явного дублирования с различными префиксами), а требует перекомпиляции.

Начальная геометрия для библиотеки `p4est` — это базовая сетка, которая состоит из набора четырехугольных (в 2D) или гексаэдральных (в 3D) ячеек. Библиотека предоставляет несколько стандартных вариантов связности ячеек этой сетки, однако существует способ задать собственный вариант связности. Каждая ячейка базовой сетки занимает свое отдельное локальное пространство, при дальнейшем делении ячейки разбиваются пополам в каждом измерении. В рамках библиотеки не существует глобальной, для всей сетки, системы координат. Однако пользователю при работе с каждой ячейкой предоставляется информация о корне дерева (базовой ячейке), а также об уровне и локальных координатах дочерней ячейки, позволяющая на основе информации о положении и форме ячейки базовой сетки восстановить глобальные физические координаты.

С каждой терминальной ячейкой сетки может быть ассоциирована область памяти некоторого размера, в которой пользователь может хранить свои данные о ячейке — геометрические, физические и пр. Такая область памяти фиксируется библиотекой для каждой создаваемой терминальной ячейки и отпускается, когда ячейка уничтожается или перестает быть терминальной.

Общим принципом библиотеки `p4est` является то, что функции библиотеки, итерационно проходящие по объектам сетки, используют пользовательские функции обратного вызова (callback functions) для индивидуальной работы с сеточными объектами. Такие callback-функции вызываются для каждого объекта сетки, попавшего в итерации, и этим функциям передаются в том числе параметры, идентифицирующие этот объект. Кроме этих параметров, callback-функциям, как правило, передается область памяти с пользовательскими данными, ассоциированными с ячейкой/ячейками.

Библиотека содержит большой набор функций, доступных пользователю. Основными можно назвать следующие:

- `p8est_new` — построение базовой сетки,
- `p8est_iterate` — итерация по терминальным объектам сетки,
- `p8est_refine` — измельчение ячеек сетки,
- `p8est_coarsen` — огрубление сетки (объединение ячеек),
- `p8est_balance` — дополнительное измельчение сетки для восстановления связности $2:1$,
- `p8est_partition` — перераспределение ячеек между процессорами для выравнивания нагрузки,
- `p8est_ghost_new` — конструирование призрачного слоя,
- `p8est_save` — сохранение структуры сетки и ассоциированных данных в файл,
- `p8est_load` — чтение структуры сетки и ассоциированных данных из файла.

Общий принцип построения сетки с помощью библиотеки `p4est` заключается в следующем. Октосетка строится на основе базовой сетки в области, представляющей собой прямоугольник (или параллелепипед в 3D), разбитый на заданное по каждой оси число первичных прямоугольных/параллелепипедных

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

ячеек. Эти ячейки соответствуют нулевому уровню деления. Далее некоторые ячейки нулевого уровня могут быть разделены на 4 (или 8) ячеек, соответствующих первому уровню деления, ячейки первого уровня могут быть разделены на ячейки второго уровня и т.д. Библиотека `p4est` в своей внутренней структуре для каждой ячейки базовой сетки хранит отдельное квадро- или октодерево, описывающее выполненный процесс деления. Ячейки самого высокого уровня деления, соответствующие листьям этих деревьев, называются терминальными. Совокупность деревьев, относящихся к ячейкам базовой сетки, далее будем называть лесом. Ниже будет описан опыт применения библиотеки `p4est` для построения октосетки в области сложной структуры.

3. Управляемое построение октосетки. В этом разделе кратко описан реализованный автором алгоритм, предназначенный для управляемого построения октосетки. Реализация именно этого алгоритма стала стимулом поиска низкоуровневой библиотеки с достаточно развитым функционалом, способной выполнять основные действия над октосеткой. Целью алгоритма является построение адаптированных октосеток для областей со сложной границей, в том числе — не выпуклых и не односвязных. Ниже кратко будут перечислены основные идеи алгоритма. Остальная часть работы будет посвящена описанию того, как функции библиотеки `p4est` могут помочь в их реализации.

Процесс построения сетки заключается в последовательном управляемом выполнении ряда действий. Сначала строится базовая сетка в объемлющем прямоугольнике/параллелепипеде. Затем указываются ячейки, которые должны быть разделены на 4 (или 8). Этот процесс выделения некоторого подмножества ячеек (только терминальных) и их деления на 4/8 ячеек следующего уровня повторяется до тех пор, пока не будет построена требуемая сетка. Так как расчетная область в общем случае не является прямоугольной (или параллелепипедной), а библиотека `p4est` работает только с топологически прямоугольными (или параллелепипедными) областями, полностью заполненными ячейками, некоторые ячейки могут оказаться вне рабочей области. Поэтому наряду с операцией деления выбранных ячеек предусмотрена операция пометки ячеек признаком “внешняя”. Информация о таких ячейках по-прежнему хранится во внутренней структуре леса деревьев в силу специфики библиотеки, но эти ячейки уже не участвуют в дальнейших операциях деления (а затем не будут участвовать в процессе численного моделирования).

В целях универсализации, указанная выше процедура построения базовой сетки и последовательных операций деления и удаления выбранных ячеек описывается в текстовом файле. В начале файла указываются параметры базовой сетки, начинающиеся с указания размерности задачи — 3D или 2D. Затем с помощью набора команд вида “условие–действие”, где “условие” — это геометрический селектор, описывающий некоторую область ячеек, производятся указанные действия над отобранными селектором ячейками.

Для трехмерного случая параметры базовой сетки включают в себя описание параллелепипеда, внутри которого будет построена сетка. Параллелепипед задается через координаты центра и координаты трех точек в центрах граней, которым принадлежит одна общая вершина (тем самым допускается скошенный параллелепипед). Эти точки определяют три ортогональных направления, на которые ориентируются другие параметры. Сетка задается либо через число ячеек по каждому из направлений, либо по примерному размеру ячеек. В последнем случае число ячеек по каждому направлению будет задано так, чтобы в наилучшей степени приблизить реальный размер базовых ячеек к указанному. Кроме того, для каждого из направлений указывается признак периодичности сетки.

В 3D реализованы селекторы со следующими видами геометрических объектов:

- параллелепипед — через координаты центра и координаты трех точек в центрах граней, которым принадлежит одна общая вершина;
- сфера — через координаты центра и радиус;
- цилиндр — через координаты двух точек в центрах оснований и радиус;
- конус (возможно — усеченный) — также через координаты центров оснований и два радиуса (один радиус может быть нулевым);
- произвольная STL-поверхность — через имя файла.

Для этих геометрических объектов в селекторах указывается:

- одно из отношений “около”, “внутри” или “снаружи” (последние два при использовании с STL требуют замкнутости поверхности);
- расстояние, определяющее ленту около поверхности объекта, внутри которой, с учетом указанного отношения, выбираются ячейки (не указанное расстояние означает бесконечность);

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

- ограничение на текущий уровень деления ячейки — допустимо ограничение “равно”, “меньше или равно” или “больше или равно” заданной константе (не указанное ограничение означает отсутствие ограничения);
- способ отбора ячейки — по попаданию в выбранную область центра ячейки, всех вершин ячейки или хотя бы одной из вершин ячейки.

Аналогичным образом задаются базовая сетка и селекторы в двумерном случае. Для двумерного случая определены следующие геометрические объекты: параллелограмм, окружность, трапеция/треугольник и ломаная линия.

Как отмечалось выше, в реализации автора процесс последовательного построения октосетки описывается в текстовом файле-задании, каждая строка которого соответствует одной операции по уточнению октосетки. Производится последовательное чтение строк текста, их разбор и выполнение описанных действий над ячейками, удовлетворяющими указанным геометрическим ограничениям. Результатом обработки всех строк задания является файл в формате `p4est`. Реализация разбора файла с заданием на построение сетки ниже описана не будет. Дальнейшее изложение сосредоточено на применении библиотеки `p4est` для реализации действий, выполняющихся при построении квадро- или октосетки.

4. Использование библиотеки `p4est`.

4.1. Инициализация внутренней структуры. Перед изложением примеров использования библиотеки следует обратить внимание на возможную путаницу в обозначениях. Библиотека называется `p4est`. Префиксом функций может быть либо `p4est_`, либо `p8est_` в зависимости от размерности сетки (в ряде специальных случаев используется также префикс `p6est_`). Типы данных имеют тот же префикс. Глобальная переменная, обеспечивающая доступ к лесу деревьев, также называется `p4est` или `p8est` и имеет тип, соответственно, `p4est_t` или `p8est_t`. Ниже все примеры кода будут приведены для трехмерного случая, поэтому в большинстве случаев используется префикс `p8est_`.

Инициализация среды библиотеки `p4est` — это процесс, который готовит MPI-инфраструктуру, внутренние данные и режимы отладки библиотеки. В листинге 1 приведена минимальная корректная последовательность команд языка C++, обеспечивающая правильную распределенную работу функций библиотеки.

Листинг 1. Инициализация и финализация библиотек в MPI-окружении
Listing 1. Initialization and finalization of libraries in MPI environment

```
1  MPI_Init(&argc, &argv);
2  sc_init(MPI_COMM_WORLD, 1, 1, NULL);
3  p8est_init(NULL, SC_LP_DEFAULT);
4
5  ... работа с p8est ...
6
7  p8est_finalize();
8  sc_finalize();
9  MPI_Finalize();
```

Здесь производится инициализация среды MPI, затем в MPI-режиме инициализируется служебная библиотека SC (Scientific Computing). Аргументы имеют следующие типовые значения (вместо булевых типов библиотека использует целочисленные со значениями 0 или 1):

```
sc_init(
  MPI_COMM_WORLD,                 // коммуникатор
  1,                              // перехват сигналов (SIGSEGV и др.)
  1,                              // печать backtrace при аварии
  NULL                            // лог-файл (NULL → stderr)
);
```

Инициализация библиотеки `p4est` производится функцией

```
p8est_init(log_handler, log_threshold);
```

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Здесь:

- `log_handler` — адрес пользовательской функции логирования или `NULL` для использования стандартной лог-функции SC;
- `log_threshold` — уровень логирования, перечисленный в типе `sc_log_priority_t`; часто используемые значения — `SC_LP_DEFAULT` (стандартный уровень), `SC_LP_ERROR` или `SC_LP_SILENT`.

Указанная функция регистрирует типы `p4est` в служебных данных SC, подготавливает свои глобальные структуры, но не создает лес и связность между деревьями.

4.2. Построение базовой сетки. Перед созданием леса деревьев необходимо описать топологию сетки. Существует несколько реализованных топологий, и можно создать собственную топологию. В описываемом в настоящей статье модуле построения сетки использована ijk-топология, которая создается функцией

```
p8est_connectivity_t* p8est_connectivity_new_brick (
  int nx, int ny, int nz,                          // число ячеек по направлениям
  int periodic_x, int periodic_y, int periodic_z   // 0 или 1: признак периодичности сетки
);
```

Указанная функция, как настаивает описание, применяется только для области, являющейся прямым параллелепипедом. Однако, так как библиотека не оперирует физическими координатами, учитывая только связность ячеек, эта функция вполне пригодна для инициализации топологии в скошенных параллелепипедах и вообще в любых областях, которые гомеоморфны прямому параллелепипеду. Результатом этой функции является адрес структуры данных типа `p8est_connectivity_t`, которую в конце выполнения программы следует освободить (см. пример ниже).

После того как построена топология, она может быть использована для построения базовой сетки при помощи функции `p8est_new` (или ее более продвинутого аналога `p8est_new_ext`)

```
p8est_t* p8est_new (
  MPI_Comm mpicomm,
  p8est_connectivity_t* connectivity,
  size_t data_size,
  p8est_init_t init_fn,
  void *user_pointer
);
```

Назначение параметров функции следующее.

- `MPI_Comm mpicomm` — MPI-коммуникатор. Например, `MPI_COMM_WORLD`.
- `p8est_connectivity_t* conn` — построенная предварительно топология.
- `int data_size` — указывает размер в байтах области, которая будет выделяться и ассоциироваться для каждой терминальной ячейки. Например, если есть необходимость ассоциировать с ячейкой тройку координат, то `data_size = 3 * sizeof(double)`. Если указать `data_size=0`, то память выделяться не будет.
- `void* init_fn` — пользовательская callback-функция. Эта функция будет вызвана для каждой создаваемой ячейки после выделения ассоциированной памяти. Служит для инициализации структуры этой памяти. Сигнатуру и пример использования см. ниже.
- `void* user_pointer` — произвольный указатель, который без изменения будет передан в callback-функцию через ее аргумент. Такая возможность обычно не используется, так как все необходимые данные можно передать в функцию через статические или внешние переменные.

Функция возвращает указатель на локальный (принадлежащий данному процессору) лес деревьев. Этот лес будет в дальнейшем использоваться во всех других функциях.

Функция построения связности (например, `p8est_connectivity_new_brick`) и функция построения базовой сетки резервируют под свои нужды некоторые участки памяти, которые в конце программы следует освободить (см. листинг 2).

Callback-функция `init_fn` должна быть описана следующим образом:

```
void init_fn(
  p8est_t* p8est,                  // указатель на лес
```

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Листинг 2. Пример создания базовой сетки и освобождения соответствующих ресурсов при завершении программы

Listing 2. An example of creating a base grid and freeing up the corresponding resources when the program ends

```
p8est_connectivity_t* conn =
  p8est_connectivity_new_brick (n0.x, n0.y, n0.z, 0, 0, 0);
p8est_t* p8est = p8est_new(MPI_COMM_WORLD, conn, sizeof(Cell), init_fn, NULL);

// построение октосетки

p8est_destroy(p8est);
p8est_connectivity_destroy(conn);
```

Листинг 3. Пример callback-функции инициализации данных ячейки

Listing 3. Example of callback function for cell's data initialization

```
void init_fn(p8est_t* p8est, p4est_topidx_t which_tree,
             p8est_quadrant_t* quadrant) {
  // логический индекс (I,j,k) дерева в лесе размером (n0.x, n0.y, n0.z):
  int i = which_tree % n0.x;
  int j = (which_tree / n0.x) % n0.y;
  int k = which_tree / (n0.x * n0.y);
  // уровень деления ячейки, для базовой ячейки lev = 0:
  int lev = quadrant->level;
  // ассоциированная структура данных:
  Cell* cell = (Cell*)q->p.user_data;
  // размер ячейки:
  cell->hx = (L.x / n0.x) / (1 << lev);
  cell->hy = (L.y / n0.y) / (1 << lev);
  cell->hz = (L.z / n0.z) / (1 << lev);
  // локальные координаты ячейки в дереве:
  double ox = (quadrant->x + 0.5) * cell->hx;
  double oy = (quadrant->y + 0.5) * cell->hy;
  double oz = (quadrant->z + 0.5) * cell->hz;
  // физические координаты центра ячейки:
  cell->pos.x = orig.x + i * (L.x / n0.x) + ox;
  cell->pos.y = orig.y + j * (L.y / n0.y) + oy;
  cell->pos.z = orig.z + k * (L.z / n0.z) + oz;
}
```

```
  p4est_topidx_t which_tree,       // индекс дерева, которому принадлежит ячейка
  p8est_quadrant_t* quadrant       // информация о ячейке
);
```

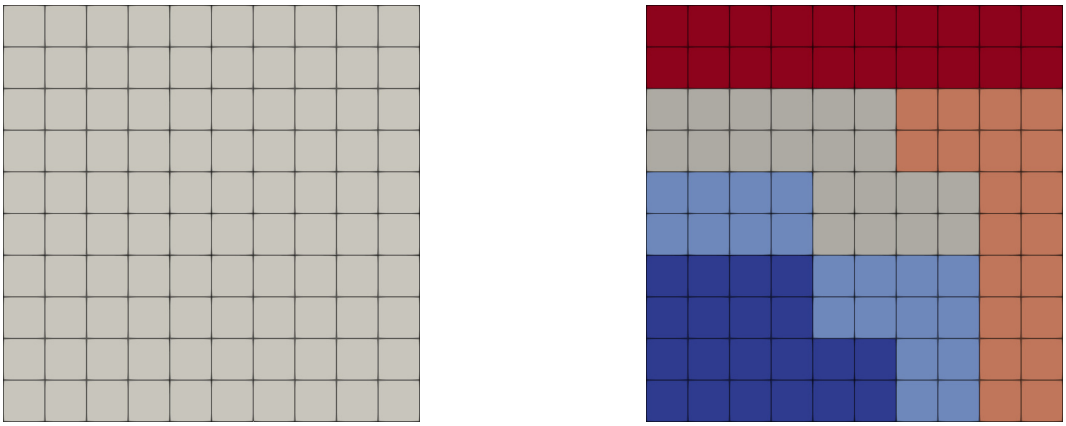
В листинге 3 приведен пример callback-функции инициализации данных ячейки для простого случая — базовой сетки размером `n0.x, n0.y, n0.z` ячеек в прямоугольном параллелепипеде с началом в `(orig.x, orig.y, orig.z)`, соосным системе координат, с длиной сторон `L.x, L.y, L.z`. Здесь был использован тот факт, что деревья в лесу индексируются по формуле

$$\texttt{which\_tree} = \texttt{i} + \texttt{n0.x} * (\texttt{j} + \texttt{n0.y} * \texttt{k}).$$

Пример базовой квадросетки приведен на рис. 1. На этом рисунке и следующих, для наглядности, будет приведена сетка для двумерного случая. На рис. 1 a показаны ячейки базовой сетки, на рис. 1 b цветовой картой показано распределение ячеек по процессорам при запуске на 5 процессорах.

4.3. Процесс измельчения сетки. После построения базовой сетки в соответствии с командами в файле задания выполняется одна или несколько массовых операций деления и/или удаления терми-

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212



a) b)

Рис. 1. Базовая сетка (2D) размером 10×10 ячеек: a) ячейки, b) ранг процессоров

Fig. 1. Base grid (2D) with size 10×10 cells: a) cells, b) processors rank

нальных ячеек. В данном разделе будет описано применение функций библиотеки `p4est` для выполнения деления ячеек (измельчения сетки). Деление всегда производится без внутренних итераций. Таким образом, каждая ячейка, удовлетворяющая геометрическому условию команды, однократно делится на 8 (или 4 в 2D) меньших ячеек следующего уровня. Для того чтобы измельчить сетку еще раз, в файле-задании необходимо написать две команды.

В квадро- и октосетках поддерживается принцип 2 : 1 (хотя для случая 3D он выглядит как 4 : 1), согласно которому уровень деления соседних ячеек не должен отличаться больше чем на единицу. При выполнении такого принципа к одной ячейке квадросетки может примыкать одна равновеликая ячейка или 2 ячейки размером в 2 раза меньше (отсюда — название принципа 2 : 1). Для октосеток в трехмерном случае это соответствует либо примыканию одной равновеликой ячейки, либо примыканию 4 ячеек меньшего размера.

Обработка одной команды деления из файла-задания осуществляется следующей последовательностью функций библиотеки:

- `p8est_refine` — функция разбиения ячеек;
- `p8est_balance` — функция дополнительного разбиения ячеек для обеспечения принципа 2 : 1;
- `p8est_partition` — функция перераспределения ячеек между процессорами с целью выравнивания нагрузки.

Данная последовательность функций выполняется для каждой команды деления в файле-задании. Рассмотрим более подробно использование этих функций.

Функция уточнения сетки `p8est_refine` вызывается со следующими параметрами:

```
p8est_refine(p8est, 0, refine_fn, replace_fn);
```

где

- `p8est` — лес деревьев;
- `0` — признак, означающий “не делать рекурсивного уточнения ячейки”;
- `refine_fn` — callback-функция, вызывающаяся для каждой терминальной ячейки и определяющая, нужно ли ее делить;
- `replace_fn` — callback-функция, вызывающаяся только для разбиваемых ячеек (если `refine_fn` возвратила 1) для инициализации данных в новых ячейках.

Рассмотрим две callback-функции, использующиеся как аргументы в `p8est_refine`. Первая функция имеет следующую сигнатуру:

```
int refine_fn(
  p8est_t *p8est,                  // лес
  p4est_topidx_t which_tree,       // дерево
```

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

```
  p8est_quadrant_t *q              // квадрант/октант
);
```

Функция вызывается для каждой терминальной ячейки сетки и имеет те же аргументы, что и рассмотренная выше функция инициализации `init_fn`. Ассоциированную с ячейкой структуру можно получить следующим образом:

```
Cell* cell = (Cell*)q->p.user_data;
```

Функция должна возвратить 1, если ячейку надо разделить на меньшие ячейки, или 0, если не надо. В случае, если для какой-то ячейки было принято решение об измельчении, то сразу же вызывается callback-функция `replace_fn`, указанная в последнем параметре функции `p8est_refine` и имеющая следующую сигнатуру:

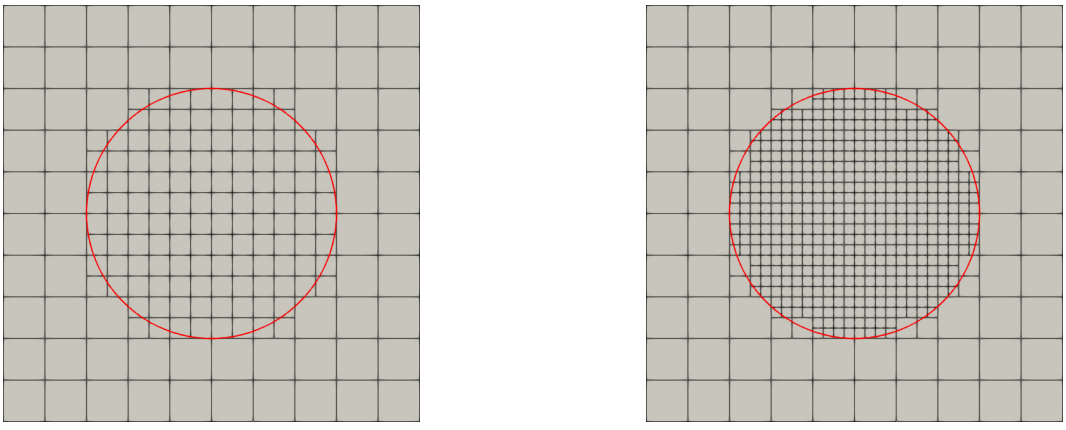
```
void replace_fn(
  p8est_t* p8est,                  // лес
  p4est_topidx_t which_tree,       // дерево
  int n_out,                       // количество замещающих ячеек
  p8est_quadrant_t* out[],         // замещающие ячейки
  int n_in,                        // количество замещаемых ячеек
  p8est_quadrant_t* in[]           // замещаемые ячейки
);
```

Вообще говоря, эта функция может быть использована как для измельчения, так и для огрубления сетки — на основе проверки количества замещаемых и замещающих ячеек. В данном контексте она применяется для измельчения, поэтому число замещаемых ячеек всегда равно 1, а число замещающих равно 8 в 3D или 4 в 2D. Соответственно, через массив `in[]` передается один указатель, а через массив `out[]` — 8 или 4. Получить ссылку на ассоциированную структуру можно следующим образом:

```
Cell* oldCell = (Cell*) in[0]->p.user_data;
for (int i = 0; i < P8EST_CHILDREN; i++) {
  Cell* newCell = (Cell*) out[i]->p.user_data;
  ...
}
```

Библиотека самостоятельно выделяет участки памяти для всех замещающих ячеек перед вызовом callback-функции `p8est_refine` и освобождает память, ассоциированную с замещаемой ячейкой, после выхода из этой функции.

Результат однократного и двукратного измельчения области сетки внутри окружности приведен на рис. 2 a, b. Видно, что после двукратного измельчения принцип 2 : 1 нарушен. Восстановление принципа 2 : 1 производится с помощью вызова функции



a) b)

Рис. 2. Квадросетка после измельчения ячеек внутри окружности: a) однократного; b) двукратного

Fig. 2. Quad mesh after refinement of cells inside the circle: a) single; b) double

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

```
void p8est_balance(
  p8est_t* p8est,
  p8est_connect_type_t btype,
  p8est_replace_t replace_fn
);
```

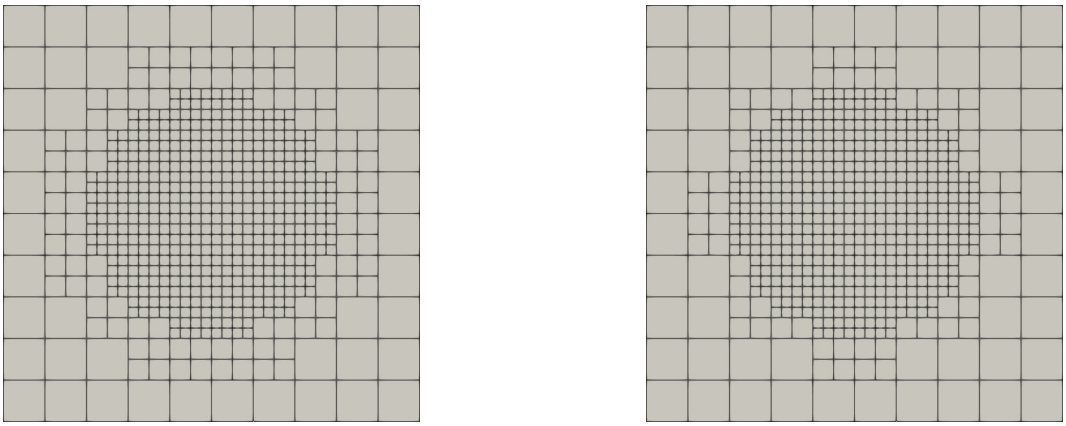
где параметры имеют следующий смысл:

- p8est — указатель на лес;
- btype — тип топологической связности, по которой обеспечивается правило 2 : 1:
  - `P8EST_CONNECT_FACE` — проверка только по граням;
  - `P8EST_CONNECT_EDGE` — по граням и ребрам;
  - `P8EST_CONNECT_CORNER` — по граням, ребрам и узлам;
  - `P8EST_CONNECT_FULL` — эквивалент CORNER, для корректного обеспечения принципа 2 : 1 можно использовать `btype` = `P8EST_CONNECT_FULL`;
- `replace_fn` — callback-функция, инициализирующая структуры, ассоциированные с новыми ячейками.

Принцип действия функции `p8est_balance` аналогичен принципу действия рассмотренной выше функции `p8est_refine` за тем исключением, что в `p8est_refine` для принятия решения о необходимости измельчения ячейки используется callback-функция, а в `p8est_balance` принятие такого решения производится самой библиотекой в области нарушения принципа 2 : 1. При этом, обычно, в этих двух функциях используется одна и та же callback-функция инициализации данных ассоциированной структуры.

На рис. 3 a, b для сетки, полученной выше двукратным измельчением ячеек, попавшим внутрь окружности (см. рис. 2 b), приведен результат работы функции `p8est_balance`, восстанавливающий принцип смежности 2 : 1. На рис. 3 a приведен результат выполнения функции `p8est_balance` с типом связности `P8EST_CONNECT_FULL`, а на рис. 3 b — с типом связности `P8EST_CONNECT_EDGE`. Причина различия в том, что функция `p4est_balance` для каждой ячейки ищет соседние, для которых уровень деления как минимум на два меньше. Если находит, то эти соседние (с меньшим уровнем деления) делит. Потом этот алгоритм повторяется до тех пор, пока уровни деления смежных ячеек не будут совпадать или отличаться на единицу. Параметр `btype` функции `p4est_balance` определяет, какие ячейки являются соседними. Константа `P4EST_CONNECT_CORNER` (или ее аналог `P4EST_CONNECT_FULL`) задает соседство при общих гранях, ребрах или узлах. А константа `P4EST_CONNECT_EDGE` говорит о том, что при определении соседства узлы рассматривать не надо. Это и определяет разницу в результате дополнительного измельчения сетки. Ситуация в трехмерном случае аналогичная.

Процесс измельчения сетки, как правило, затрагивает не всю расчетную область, поэтому где-то количество ячеек возрастет в 8 (или 4) раз, где-то останется прежним. Эта неравномерность может при-

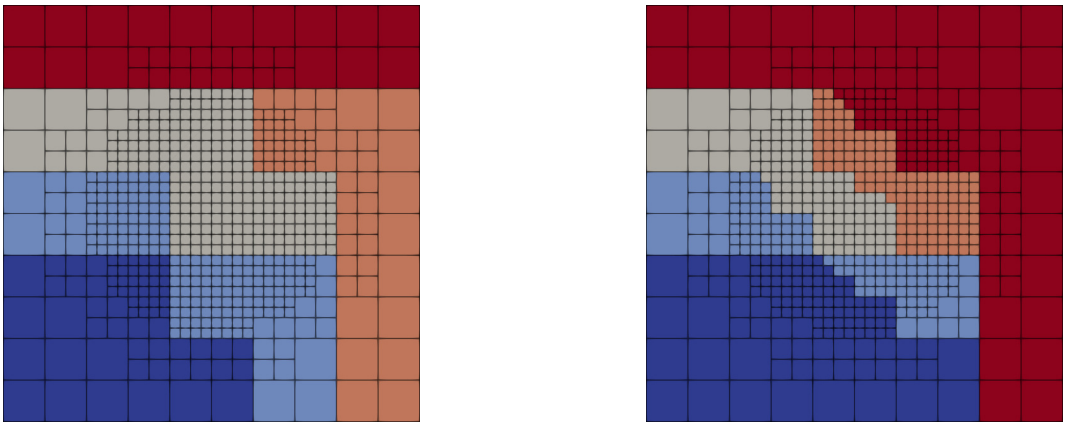


a) b)

Рис. 3. Результат восстановления принципа связности 2 : 1: a) `P8EST_CONNECT_FULL`; b) `P8EST_CONNECT_EDGE`

Fig. 3. Result of balancing 2 : 1: a) `P8EST_CONNECT_FULL`; b) `P8EST_CONNECT_EDGE`

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212



a) b)

Рис. 4. Результат выравнивания нагрузки: a) до выравнивания; b) после выравнивания

Fig. 4. Result of load balancing: a) before balancing; b) after balancing

Таблица 1. Количество ячеек в партициях до и после выравнивания

Table 1. Number of cells in partitions before and after balancing

Количество ячеек у процессора Number of cells per processor	0	1	2	3	4
до выравнивания before balancing	68	185	239	68	32
после выравнивания after balancing	118	118	119	118	119

вести к тому, что нагрузка на процессоры в случае многопроцессорной обработки станет неравномерной. Библиотека `p4est` обладает встроенной возможностью выравнивать нагрузку путем передачи ячеек от одного процессора другому. Это делается с помощью функции

```
void p8est_partition(
  p8est_t* p8est,
  int allow_for_coarsening,
  p8est_weight_t weight_fn
);
```

Назначение параметров здесь следующее:

- `p8est` — указатель на лес квадро- или октодеревьев;
- `allow_for_coarsening` — (0 или 1) позволять ли библиотеке проводить огрубление сетки по своему усмотрению;
- `weight_fn` — `NULL` или callback-функция, возвращающая вес (вычислительную сложность) ячейки.

Как правило, автоматическое огрубление не используется, так как в библиотеке существует соответствующая функция для управляемого объединения ячеек. Если в третьем параметре указать `NULL`, то считается, что вес каждой ячейки равен 1.

Рис. 4 демонстрирует результат применения функции `p8est_partition` выравнивания нагрузки на процессоры. При запуске демонстрационной программы на 5 процессорах распределение ячеек базовой сетки показано на рис. 1 b. Результат измельчения этой сетки приведен на рис. 4 a, результат выравнивания — на рис. 4 b. Количество ячеек на процессорах до и после выравнивания приведено в табл. 1.

Листинг 4, суммируя сказанное выше, демонстрирует процесс измельчения ячеек сетки.

4.4. Процесс отбрасывания ненужных ячеек. Библиотека `p4est` не предусматривает реального удаления объектов из квадро- или октодеревьев. Поэтому, чтобы сохранить возможность работать с сеткой с границей сложной структуры, приходится вместо удаления ячеек, лежащих вне расчетной области,

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Листинг 4. Применение функций процесса измельчения сетки
Listing 4. Usage of functions in refinment process

```
for(int i=0; i<nInstr; i++) {           // цикл по инструкциям файла задания
  command = commands[i];                // текущая команда из файла задания
  if(command->action==CA_DIVIDE) {      // если это команда измельчения
    p8est_refine(p8est, 0, FnCheckRefine, replace_fn);     // измельчение
    p8est_balance(p8est, P8EST_CONNECT_FULL, replace_fn); // 2\,:\,1
    p8est_partition(p8est, 0, NULL);    // выравнивание нагрузки
  } else {
    ...
  }
}

// проверка: надо ли мельчить ячейку
int FnCheckRefine(p8est_t* p8est, p4est_topidx_t which_tree,
                                          p8est_quadrant_t* q) {
  Cell* cell = (Cell*)q->p.user_data;
  if(cell->IsSatisfied(command))        // если ячейка удовлетворяет условию
    return 1;                           // она будет измельчена
  return 0;                             // без измельчения
}

// инициализация параметров новых маленьких ячеек
void FnReplaceRefine(p8est_t* p8est, p4est_topidx_t which_tree,
  int n_new, p8est_quadrant_t* news[],  // замещающие ячейки
  int n_old, p8est_quadrant_t* olds[]   // замещаемые ячейки
) {
  if(n_old==1) {                        // это измельчение
    assert(n_new==P8EST_CHILDREN);
    Cell* oldCell = (Cell*) olds[0]->p.user_data;
    for (int i = 0; i < P8EST_CHILDREN; i++) {
      Cell* newCell = (Cell*) news[i]->p.user_data;
      // инициализация данных в новых ячейках
    }
  } else {                              // это укрупнение
    assert(n_old==P8EST_CHILDREN && n_new==1);
  }
}
```

помечать их как “удаленные”. В процессе построения сетки отмеченные этим признаком ячейки не участвуют в отборе по геометрическим параметрам, не измельчаются функцией `p8est_refine`, однако могут быть разбиты на более мелкие при восстановлении принципа связности 2:1 функцией `p8est_balance`. Из сказанного выше следует, что процесс обработки инструкции удаления ячеек сводится к проходу по всем терминальным “живым” (не помеченным признаком “удаленная”) ячейкам и присвоению тем из них, которые удовлетворят соответствующему геометрическому ограничению, признака “удалена”. Для этого используется функция итераций по ячейкам со следующей сигнатурой:

```
void p8est_iterate(
  p8est_t* p8est,
  p8est_ghost_t* ghost,
  void* user_data,
  p8est_iter_volume_t volume_fn,
  p8est_iter_face_t face_fn,
  p8est_iter_edge_t edge_fn,
  p8est_iter_corner_t corner_fn
);
```

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Назначение параметров здесь следующее:

- `p8est` — лес деревьев, по объектам которого выполняется обход;
- `ghost` — призрачный слой или `NULL`;
- `user_data` — произвольный указатель, передающийся в callback-функции, или `NULL`;
- `volume_fn` — callback-функция для итераций по ячейкам или `NULL`;
- `face_fn` — callback-функция для итераций по граням или `NULL`;
- `edge_fn` — callback-функция для итераций по ребрам или `NULL`;
- `corner_fn` — callback-функция для итераций по узлам или `NULL`.

При использовании этой функции необходимо учитывать особенности ее поведения при работе в среде нескольких MPI-потоков. Ячейки сетки распределены некоторым образом среди MPI-процессоров. Далее фрагмент сетки, объекты которой хранятся на одном процессоре, будем называть партицией.

Если в одном вызове `p8est_iterate` указать callback-функции для нескольких типов объектов, то библиотека гарантирует следующее:

- первым будет вызов callback-функции для ячейки;
- вызов callback-функции для грани произойдет только после того, как выполнятся вызовы функций для всех ячеек, прилегающих к этой грани — *но только ячеек, принадлежащих данной партиции*;
- вызов callback-функции для узла произойдет только после вызова функций для всех граней, сходящихся к этому узлу.

С другой стороны:

- не гарантируется, что перед вызовом callback-функции для узла будут вызваны callback-функции для всех 6 граней всех 8 ячеек, которым принадлежит узел (количества здесь приведены для 3D);
- не производятся вызовы callback-функций для призрачных ячеек (см. оговорку выше);
- не производятся вызовы callback-функций для других объектов (граней, ребер и узлов), если у них нет смежных ячеек, принадлежащих данной партиции.

Обход ячеек осуществляется в порядке возрастания индекса Мортона [13], поочередно для всех деревьев леса, принадлежащих данной партиции.

Сигнатуру callback-функций в `p8est_iterate` можно найти в сопроводительной документации. Здесь будут даны пояснения по callback-функции для итераций по ячейкам `volume_fn`:

```
typedef void (*p8est_iter_volume_t)(
  p8est_iter_volume_info_t* info,
  void* user_data
);
```

Второй аргумент `user_data` — это адрес, указанный в `p8est_iterate`. Основная информация о ячейке передается через аргумент `info`:

```
typedef struct p8est_iter_volume_info {
  p8est_t*          p8est;           // лес
  p8est_ghost_t*    ghost_layer;     // призрачный слой
  p8est_quadrant_t* quad;            // квадрант
  p4est_locidx_t    quadid;          // id квадранта в массиве деревьев
  p4est_topidx_t    treeid;          // id дерева, содержащего квадрант
} p8est_iter_volume_info_t;
```

Адрес данных, ассоциированных с ячейкой, можно получить следующим образом:

```
  p8est_quadrant_t* q = info->quad;
  Cell* cell = (Cell*) q->p.user_data;
```

Callback-функции для других объектов включают кроме топологической информации также информацию из нескольких смежных ячеек. При многопроцессорном выполнении такие ячейки могут принадлежать другой партиции и данные для них надо получать через MPI-обмены. Для межпроцессорных коммуникаций служит призрачный слой (см. параметр `ghost`).

В настоящей статье описывается процесс построения сетки, в котором итерации по граням, ребрам и узлам не выполняются. Соответственно, призрачный слой не строится и не используется.

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Полностью процедура отметки ячеек вне рабочей области в программе построения приведена в листинге 5.

Листинг 5. Итерации по ячейкам для простановки флага “удалена”
Listing 5. Iteration throw cells for flag “deleted” setting

```
p8est_iterate(p8est, NULL, NULL, cut_fn, NULL, NULL, NULL);

// Callback функция для отметки удаленных ячеек
static void cut_fn(p8est_iter_volume_info_t* info, void* userData) {
  p8est_quadrant_t* q = info->quad;
  Cell* cell = (Cell*) q->p.user_data;
  if(selector->Select(cell))
    cell->status |= CELL_DELETED;
}
```

После того как некоторые ячейки помечены флагом “удалена”, может сложиться ситуация, когда все 8 (или 4 в 2D) ячеек, полученных из одной в результате предыдущих операций измельчения, окажутся со статусом “удалена”. В этом случае можно снова объединить их в одну, уменьшив общее количество ячеек в сетке. Этим целям служит функция

```
void p8est_coarsen(
  p8est_t* p8est,                 // указатель на лес
  int coarsen_recursive,          // нужно ли использовать рекурсию
  p4est_coarsen_t coarsen_fn,     // callback-функция принятия решения
  p4est_init_t init_fn            // callback-функция инициализации данных в ячейках
)
```

Объединиться могут не любые ячейки, а только те, которые имеют единого родителя в окто- (или квадро-) дереве, т.е. те, которые ранее были получены делением одной ячейки. В этом смысле действие функции `p8est_coarsen` является попыткой отменить действие функции `p8est_refine`, рассмотренной в предыдущем разделе. Если рекурсия не используется (`coarsen_recursive=0`), то происходит однократное слияние 8 (или 4) ячеек в одну. При включенной рекурсии библиотека после объединения попробует объединить новые более крупные ячейки еще раз.

Так же, как и `p8est_refine`, функция `p8est_coarsen` пробегает по всем ячейкам, потенциально пригодным для объединения (т.е. имеющим одного родителя в дереве) и вызывает callback-функцию `coarsen_fn` для принятия решения о возможности объединения. Для этого функция `coarsen_fn` получает через аргументы массив из 8 (или 4) ячеек, которые предлагается объединить. Если `coarsen_fn` возвратит 1, то библиотека построит одну новую ячейку бóльших размеров, для которой выделит память для ассоциированной структуры данных и вызовет callback-функцию `init_fn` для инициализации этой структуры. После этого библиотека освободит память, ассоциированную с объединяемыми ячейками, и удалит их (см. листинг 6).

Здесь можно отметить, что аргументы callback-функции `FnReplaceCoarsen` совпадают с аргументами рассмотренной выше callback-функции `FnReplaceRefine`. Отличие в том, что в `FnReplaceCoarsen` через аргументы передается 8 (или 4) замещаемых ячеек и одна замещающая, а в `FnReplaceRefine` наоборот — одна замещаемая и 8 (или 4) замещающих. Так как количество тех и других также передается через аргументы, в обоих случаях можно использовать одну и ту же функцию.

Есть важное недокументированное обстоятельство, которое надо учитывать при огрублении (объединении) ячеек. Рассмотрим некоторую ячейку, которая была разбита на 8 (или 4) дочерних ячеек. Если в результате выравнивания нагрузки функцией `p8est_partition` эти дочерние ячейки оказались в разных партициях (были распределены по разным процессорам), то они уже не смогут быть объединены в одну. Это ограничение библиотеки. На рис. 5 приведены результаты попытки объединить ячейки сетки, лежащие в нижней левой четверти сетки, до исходного состояния (до размеров базовой сетки). Цветом обозначено распределение ячеек по партициям (0–4). На рис. 5 a показан результат огрубления при исходном разбиении по партициям, показанном на рис. 4 a, а на рис. 5 b — при исходном разбиении по партициям, показанном на рис. 4 b. Можно заметить, что во втором случае не удалось огрубить ячейки

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

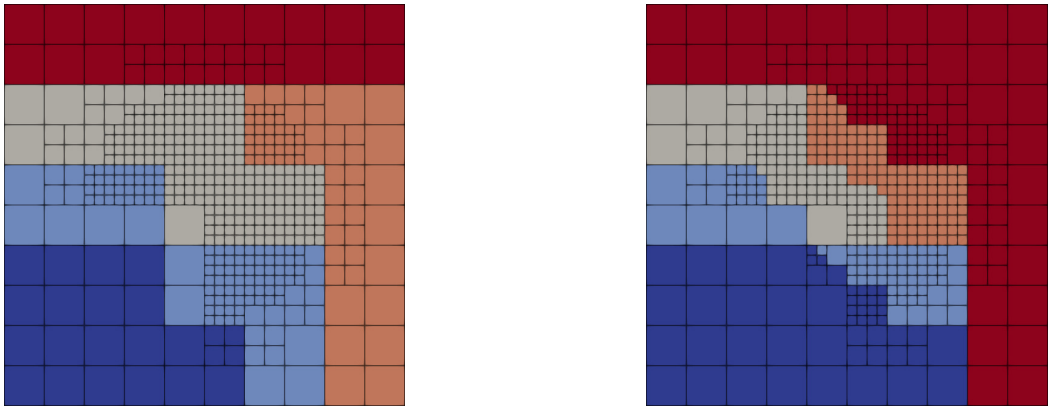
Листинг 6. Итерации по ячейкам для возможного объединения удаленных ячеек
Listing 6. Iterate over cells to possibly merge deleted cells

```
// итерации для объединения удаленных ячеек:
p8est_coarsen_ext(p8est, 1, 0, FnCheckCoarsen, NULL, FnReplaceCoarsen);

// проверка: можно ли 8 удаленных ячеек заменить на одну
int FnCheckCoarsen(p8est_t* p8est, p4est_topidx_t which_tree,
  p8est_quadrant_t* quadrants[])
{
  for(int i=0; i<P8EST_CHILDREN; i++) {
    Cell* cell = (Cell*)quadrants[i]->p.user_data;
    if((cell->status & CELL_DELETED)!=0) return 0;
  }
  return 1;
}

// инициализация параметров большой удаленной ячейки
void FnReplaceCoarsen(p8est_t* p8est,
  p4est_topidx_t which_tree,
  int nOld, p8est_quadrant_t* olds[],
  int nNew, p8est_quadrant_t* news[])
{
  assert(nOld==8 && nNew==1);
  Cell* newCell = (Cell*)news[0]->p.user_data;
  newCell->status = CELL_DELETED;
  newCell->pos = Vector::V0;
  newCell->lev = news[0]->level;

  for(int i=0; i<8; i++) {
    Cell* oldCell = (Cell*)olds[i]->p.user_data;
    newCell->pos += oldCell->pos;
  }
  newCell->pos /= 8.;
}
```

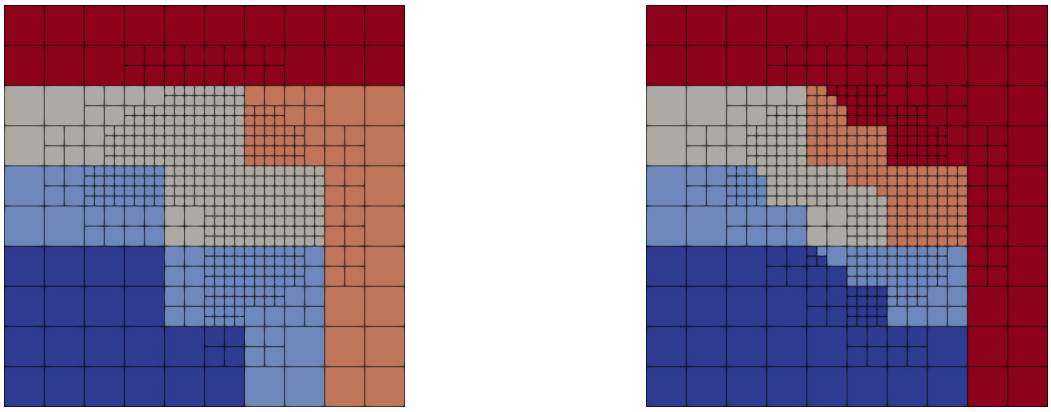


a) b)

Рис. 5. Результат огрубления ячеек в нижнем левом квадранте: a) до выравнивания нагрузки на процессоры; b) после выравнивания нагрузки на процессоры

Fig. 5. The result of coarsening the cells in the lower left quadrant: a) before balancing the load on processors; b) after balancing the load on processors

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212



a) b)

Рис. 6. Результат восстановления принципа связности 2 : 1 после огрубления: a) до выравнивания нагрузки на процессоры; b) после выравнивания нагрузки на процессоры

Fig. 6. Result of balancing 2 : 1 after coarsening: a) before balancing the load on processors; b) after balancing the load on processors

до конца, потому что, как указывалось выше, невозможно объединить ячейки, если они попали в разные партиции.

Также надо учитывать то обстоятельство, что после огрубления сетки принцип 2 : 1, скорее всего, будет нарушен и, возможно, значительно. Однако корректное выполнение итераций по граням, ребрам и узлам в функции `p8est_iterate` возможно только в случае, когда такой принцип выполнен. Поэтому после огрубления, также как и после уточнения, необходимо восстановить принцип связности 2 : 1 с помощью функции `p8est_balance`. Результат восстановления принципа 2 : 1 для двух рассмотренных выше разбиений ячеек по партициям приведен на рис. 6. На рис. 6 a приведен результат восстановления принципа 2 : 1 сетки с рис. 4 a, на рис. 6 b — сетки с рис. 4 b.

4.5. Сохранение сетки на диск. После окончания построения сетки ее следует записать в файл. Библиотека `p4est` предоставляет для этого функцию

```
void p8est_save_ext(
  const char* filename,   // имя файла
  p8est_t* p8est,         // указатель на дерево
  int save_data,          // сохранять ли ассоциированные данные
  int save_partition      // сохранять ли привязку к партициям
)
```

Параметр `save_data` указывает, надо ли вместе с топологией сетки записывать в файл ассоциированные с ячейками структуры. Параметр `save_partition`, отличный от нуля, сделает файл зависимым от числа процессоров. В этом случае не удастся прочитать файл на числе процессоров, отличном от числа процессоров в момент записи. В случае если параметр равен нулю, файл создается так, как если бы он записывался при однопроцессорном счете. Следует иметь в виду, что существующая в библиотеке функция

```
void p8est_save(
const char* filename,   // имя файла
p8est_t* p8est,         // указатель на дерево
int save_data           // сохранять ли ассоциированные данные
)
```

всегда создает файл, зависимый от числа процессоров. Функция `p8est_save_ext` (как и `p8est_save`) является коллективной и должна вызываться на всех процессорах с одинаковым значением аргументов.

Прочитать созданный файл можно в дальнейшем с помощью функции

```
p8est_t* p8est_load_ext(
  const char* filename,                // имя файла
```

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

```
  sc_MPI_Comm mpicomm,                  // MPI коммуникатор
  size_t data_size,                     // размер ассоциированной структуры
  int load_data,                        // 1: чтение ассоциированных структур
  int autopartition,                    // 1: игнорирование сохраненных партиций
  int broadcasthead,                    // 1: чтение заголовка через процессор 0
  void* user_pointer,                   // адрес, передаваемый в функцию инициализации
  p8est_connectivity_t** connectivity   // адрес указателя на структуру связности
)
```

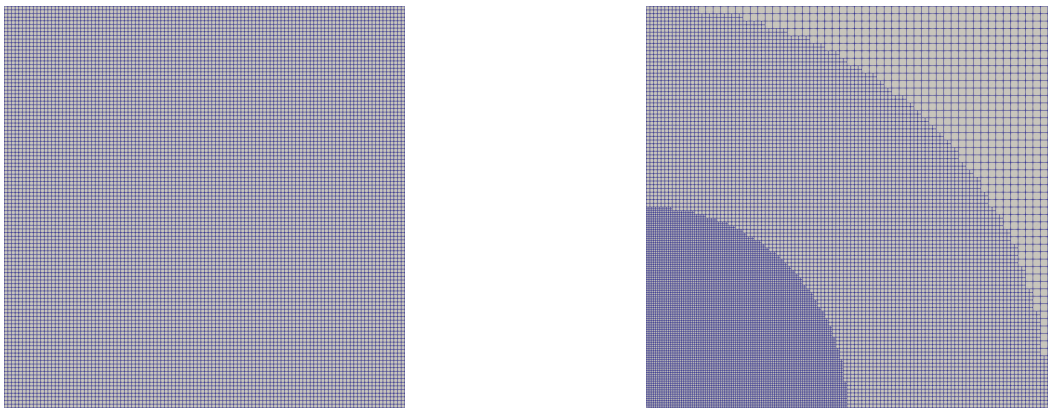
Эта функция заменяет описанную выше функцию создания базовой сетки `p8est_new`, возвращает прочитанный лес, выделяет память и заполняет прочитанными данными структуры, ассоциированные с ячейками. Упрощенный аналог `p8est_load` должен использоваться при совпадении числа процессоров в момент записи и в момент чтения, гарантируя полное совпадение записанных данных с прочитанными. Функция `p8est_load_ext` (как и `p8est_load`) является коллективной и должна вызываться на всех процессорах с одинаковым значением аргументов.

5. Оценка скорости доступа к данным в ячейках. Для оценки скорости доступа к данным в ячейках с помощью библиотеки p4est был выполнен следующий численный эксперимент. Был построен ряд сеток с различным уровнем деления. С каждой терминальной ячейкой связана некоторая структура данных, содержащая также число типа `double`. С помощью функции `p8est_iterate` выполняется проход по всем терминальным ячейкам, в каждой из которых в ассоциированных данных указанное число увеличивается на 1. Общее время выполнения функции `p8est_iterate` фиксируется, делится на число ячеек и выводится. Таким образом, данный эксперимент позволяет с некоторой степенью точности определить накладные расходы на доступ к ассоциированным данным.

Сетки для проведения численных экспериментов строились следующим образом. Базовая сетка с общим числом ячеек N строилась в единичном кубе $[0,1]^3$. Затем на этой сетке выполнялось M операций разбиения ячеек. На каждой i-й операции ($i = 0, \ldots, M-1$) разбиению подвергались ячейки, центры которых удалены от точки $(0,0,0)$ не более чем на $1/(i+1)$. Это набор концентрически вложенных шарообразных областей. Получившиеся сетки (их сечение в плоскости $y = 0$) приведены на рис. 7 и рис. 8.

Результаты экспериментов приведены в табл. 2. В каждом эксперименте число ячеек N в базовой сетке подбиралось так, чтобы после выполнения M операций разбиения общее число терминальных ячеек N_c было примерно одинаковым (около 1.3×10^6).

Время доступа к данным в ячейках оказалось слабо зависящим от уровня разбиения ячейки. Представляется, что при более реалистичной полезной нагрузке на callback-функцию итераций полученная разница во времени станет несущественной. Код, использованный в экспериментах, приведен в листингах 7 (основная часть) и 8 (callback-функции).

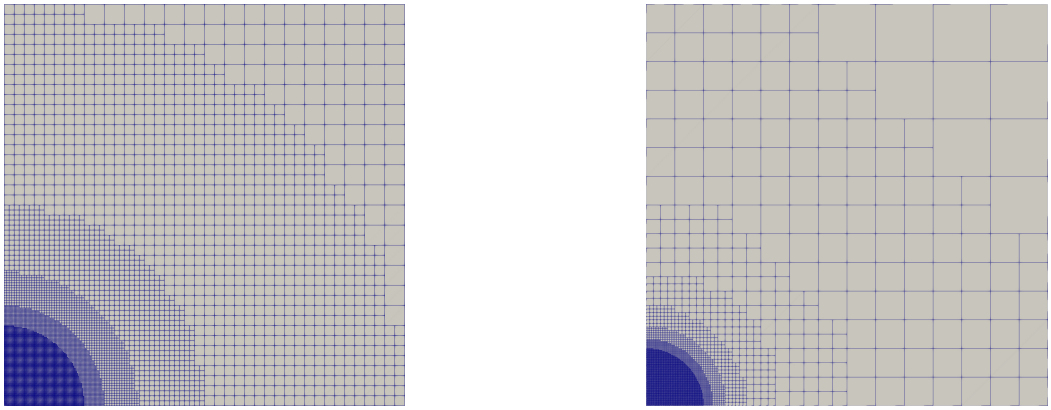


a) b)

Рис. 7. Сетки, использованные в экспериментах: a) эксперимент 1; b) эксперимент 2

Fig. 7. Grids used in experiments: a) experiment 1; b) experiment 2

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212



a) b)

Рис. 8. Сетки, использованные в экспериментах: a) эксперимент 3; b) эксперимент 4

Fig. 8. Grids used in experiments: a) experiment 3; b) experiment 4

Таблица 2. Результаты экспериментов определения скорости доступа к данным в ячейках

Table 2. Results of experiments to determine data access speed in cells

Эксперимент Experiment	N	M	N_c	мкс/ячейка us/cell
1	110	0	1331000	0.01561377
2	54	2	1311638	0.0083486854
3	20	5	1332680	0.0087044917
4	7	7	1209649	0.0095760613

Листинг 7. Код, использованный в экспериментах (основная часть)

Listing 7. Code used in the experiments (main part)

```
con=p8est_connectivity_new_brick (N, N, N, 0, 0, 0);
geom=p8est_geometry_new_connectivity(con);
p8est=p8est_new_ext(MPI_COMM_WORLD,con,1,0,0,sizeof(Cell),FnInitCell,NULL);
// построение сетки:
for(int i=0; i<M; i++) {
  dist = 1./(i+1.);
  p8est_refine_ext(p8est, 0, -1, FnCheckRefine, FnInitCell, NULL);
  p8est_balance_ext(p8est, P8EST_CONNECT_FACE, FnInitCell, NULL);
  p8est_partition_ext(p8est, 0, NULL);
}

// эксперимент:
MPI_Barrier(MPI_COMM_WORLD);
double t0 = MPI_Wtime();
p8est_iterate(p8est,NULL, NULL, FnTestCell, NULL, NULL, NULL);
MPI_Barrier(MPI_COMM_WORLD);
double t1 = MPI_Wtime();

// вывод результатов:
if(rank==0) printf("Dif-time:␣%.8g␣[mksec/cell]\n",
```

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

```
    (t1-t0)*1.e6/p8est->global_num_quadrants);
  G("");

  p8est_destroy (p8est);
  if(geom!=NULL) p8est_geometry_destroy(geom);
  if(con!=NULL) p8est_connectivity_destroy (con);
```

Листинг 8. Код, использованный в экспериментах (callback-функции)

Listing 8. Code used in the experiments (callback functions)

```
// Callback функция. Определяет необходимость разбиения ячейки
static int FnCheckRefine(p8est_t* p8est, p4est_topidx_t which_tree,
                                                    p8est_quadrant_t* q) {
  Cell* cell = (Cell*)q->p.user_data;
  double d = Vector(0.,0.,0.).GetDistanceTo(cell->pos);
  return d<dist;
}

// Callback функция присваивания значения при создании ячейки
static void FnInitCell (p8est_t* p8est, p4est_topidx_t which_tree,
                                                    p8est_quadrant_t* q) {
  Cell* cell = (Cell*)q->p.user_data;
  cell->lev = q->level;
  cell->part = prank;
  cell->SetTerminal();
  cell->G = 0;

  static const double irl = 1. / P4EST_ROOT_LEN;
  p4est_qcoord_t coords[3];                          // логические координаты
  double xyz[3];                                     // физические координаты

  p8est_quadrant_volume_coordinates(q, coords);      // центр ячейки в лог. коорд.
  p8est_geometry_transform_coordinates (geom, which_tree, coords, xyz);
  xyz[0] = xyz[0] / n0.x * bndSize.x + bndOrigin.x; // физические координаты
  xyz[1] = xyz[1] / n0.y * bndSize.y + bndOrigin.y;
  xyz[2] = xyz[2] / n0.z * bndSize.z + bndOrigin.z;

  cell->pos = Vector(xyz);                           // центр ячейки
}
void FnTestCell(p8est_iter_volume_info_t * info, void *user_data) {
  p8est_quadrant_t* q = info->quad;
  Cell* cell = (Cell*) q->p.user_data;
  cell->G = cell->G + 1.;
}
```

6. Заключение. Описан опыт использования библиотеки `p4est` для управляемого построения окто- (или квадро-) сетки в области сложной структуры. Данная задача не требовала использования некоторых интересных возможностей библиотеки. Например, не был использован механизм создания и актуализации призрачного слоя. Не были использованы итерации по граням ячеек с получением данных от смежных ячеек (количество ячеек с разных сторон от грани может быть различным, эти ячейки могут принадлежать своей партиции или лежать в призрачном слое). Также интересным было бы рассмотреть механизм динамической адаптации и огрубления сетки. И, наконец, важным является вопрос — как использовать систему хранения, предусматривающую только данные в ячейках, для, например, конечно-объемных алгоритмов, в которых данные должны сохраняться не только в ячейках, но и на гранях.

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

Листинг 9. Программа подготовки рисунков
Listing 9. Figure preparation program

```
int main (int argc, char **argv) {
  p4est_ghost_t *ghost;
  sc_MPI_Init (&argc, &argv);
  sc_MPI_Comm_size (sc_MPI_COMM_WORLD, &nproc);
  sc_MPI_Comm_rank (sc_MPI_COMM_WORLD, &prank);

  sc_init (sc_MPI_COMM_WORLD, 1, 1, NULL, SC_LP_ESSENTIAL);
  p4est_init (NULL, SC_LP_PRODUCTION);

  conn = p4est_connectivity_new_brick (10, 10, 0, 0);
  geom = p4est_geometry_new_connectivity(conn);
  p4est = p4est_new (sc_MPI_COMM_WORLD, conn, sizeof(Cell), fnInit, NULL);
  G("1a"); G("1b");

  p4est_refine (p4est, 0, refine_fn, fnInit); G("2a");   // первое деление
  p4est_refine (p4est, 0, refine_fn, fnInit); G("2b");   // второе деление
  p4est_balance(p4est, P4EST_CONNECT_FACE, fnInit); G("3a"); // 2:1 по граням
  p4est_balance(p4est, P4EST_CONNECT_FULL, fnInit); G("3b"); // 2:1 полная

  p4est_iterate(p4est,NULL,NULL,FnPart,NULL,NULL); G("4a"); // N партиции в ячейках
  if(PART) p4est_partition(p4est, 0, NULL);                 // выравнивание нагрузки
  p4est_iterate(p4est,NULL,NULL,FnPart,NULL,NULL); G("4b"); // N партиции в ячейках

  p4est_iterate(p4est, NULL, NULL, FnDel, NULL, NULL );      // отметка удаленных
  p4est_coarsen_ext(p4est,1,0,FnCoarsen,fnInit,FnReplace); G("5"); // огрубление
  p4est_balance_ext(p4est,P4EST_CONNECT_FACE,fnInit,FnReplace); G("6"); // 2:1

  p4est_destroy (p4est);
  if(geom!=NULL) p4est_geometry_destroy(geom);
  p4est_connectivity_destroy (conn);

  sc_finalize ();
  mpiret = sc_MPI_Finalize ();

  return 0;
}
```

Для подготовки иллюстраций к настоящей статье использовалась программа, главная функция которой приведена в листинге 9. Глобальная булева переменная `PART` использована для демонстрации двух вариантов — с выравниванием нагрузки на процессоры и без нее. Функция `G()` предназначена для построения графического файла с сеткой в формате VTK. Параметр функции указывает на номер рисунка в данной статье.

Автор планирует продолжить публикацию статей о возможностях библиотеки `p4est` с целью продемонстрировать ее пригодность к тому, чтобы быть основой кода численного моделирования гидродинамических течений на адаптивной октосетке.

Список литературы

1. *Meagher D.* Octree encoding: a new technique for the representation, manipulation and display of arbitrary 3-D objects by computer: techn. rep. / IPL-TR-80-111. Troy: Rensselaer Polytechnic Institute, 1980.
2. *Peskin C.S.* Flow patterns around heart valves: a numerical method // Journal of Computational Physics. 1972. **10**, N 2. 252–271. doi 10.1016/0021-9991(72)90065-4.

3. *Yang C.-H., Scovazzi G., Krishnamurthy A., Ganapathysubramanian B.* Octree-based adaptive mesh refinement and the shifted boundary method for efficient fluid dynamics simulations // Adv. Comput. Sci. Eng. 2025. **4**. 57–84. doi 10.3934/acse.2025012.
4. *Fujita K., Katsushima K., Ichimura T., et al.* Octree-based multiple-material parallel unstructured mesh generation method for seismic response analysis of soil-structure systems // Procedia Computer Science. 2016. **80**. 1624–1634. doi 10.1016/j.procs.2016.05.496.
5. *Schneiders R.* Octree-based hexahedral mesh generation // International Journal of Computational Geometry & Applications. 2000. **10**, N 04. 383–398. doi 10.1142/S021819590000022X.
6. *Burstedde C., Wilcox L.C., Ghattas O.* p4est: scalable algorithms for parallel adaptive mesh refinement on forests of octrees // SIAM Journal on Scientific Computing. 2011. **33**, N 3. 1103–1133. doi 10.1137/100791634.
7. *Isaac T., Burstedde C., Wilcox L.C., Ghattas O.* Recursive algorithms for distributed forests of octrees // SIAM Journal on Scientific Computing. 2015. **37**, N 5. C497–C531. doi 10.1137/140970963.
8. *Isaac T., Burstedde C., Ghattas O.* Low-cost parallel algorithms for 2:1 octree balance // 2012 IEEE 26th International Parallel and Distributed Processing Symposium. Shanghai: IEEE Press, 2012. pp. 426–437. doi 10.1109/IPDPS.2012.47.
9. *Burstedde C.* Parallel tree algorithms for AMR and non-standard data access // ACM Transactions on Mathematical Software. 2020. **46**, N 4. 1–31. doi 10.1145/3401990.
10. Стандартная общественная лицензия GNU (GPL), версия 2. https://www.gnu.org/licenses/old-licenses/gpl-2.0.ru.html. Дата обращения: 3 апреля 2026.
11. Исходный код библиотек p4est и sc. https://github.com/cburstedde/p4est. Дата обращения: 2 апреля 2026.
12. Меньшая стандартная общественная лицензия GNU 2.1 - Проект GNU - Фонд свободного программного обеспечения. https://www.gnu.org/licenses/old-licenses/lgpl-2.1.ru.html. Дата обращения: 3 апреля 2026.
13. *Morton G.M.* A computer oriented geodetic data base and a new technique in file sequencing: tech. rep. IBM, 1966.

Получена 24 февраля 2026 г.
Принята 23 марта 2026 г.
Опубликована 17 апреля 2026 г.

Информация об авторе

Андрей Валерьевич Соловьев — к.ф.-м.н., вед. науч. сотр.; Московский государственный университет имени М. В. Ломоносова, факультет вычислительной математики и кибернетики, Ленинские горы, 1, стр. 52, 119991, Москва, Российская Федерация.

References

1. D. Meagher, *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer: techn. rep. / IPL-TR-80-111.* (Rensselaer Polytechnic Institute, Troy, 1980).
2. C. S. Peskin, "Flow Patterns Around Heart Valves: A Numerical Method," Journal of Computational Physics **10** (2), 252–271 (1972). doi 10.1016/0021-9991(72)90065-4.
3. C.-H. Yang, G. Scovazzi, A. Krishnamurthy, and B. Ganapathysubramanian, "Octree-Based Adaptive Mesh Refinement and the Shifted Boundary Method for Efficient Fluid Dynamics Simulations," Adv. Comput. Sci. Eng. **4**, 57–84 (2025). doi 10.3934/acse.2025012.
4. K. Fujita, K. Katsushima, T. Ichimura, et al., "Octree-Based Multiple-Material Parallel Unstructured Mesh Generation Method for Seismic Response Analysis of Soil-Structure Systems," Procedia Computer Science **80**, 1624–1634 (2016). doi 10.1016/j.procs.2016.05.496.
5. R. Schneiders, "Octree-Based Hexahedral Mesh Generation," International Journal of Computational Geometry & Applications **10** (04), 383–398 (2000). doi 10.1142/S021819590000022X.
6. C. Burstedde, L. C. Wilcox, and O. Ghattas, "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees," SIAM Journal on Scientific Computing **33** (3), 1103–1133 (2011). doi 10.1137/100791634.
7. T. Isaac, C. Burstedde, L. C. Wilcox, and O. Ghattas, "Recursive Algorithms for Distributed Forests of Octrees," SIAM Journal on Scientific Computing **37** (5), C497–C531 (2015). doi 10.1137/140970963.

2026, **27** (2), 172–193. doi 10.26089/NumMet.v27r212

8. T. Isaac, C. Burstedde, and O. Ghattas, "Low-Cost Parallel Algorithms for 2:1 Octree Balance," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium* (IEEE Press, Shanghai, 2012), pp. 426–437. doi 10.1109/IPDPS.2012.47.
9. C. Burstedde, "Parallel Tree Algorithms for AMR and Non-Standard Data Access," ACM Transactions on Mathematical Software **46** (4), 1–31 (2020). doi 10.1145/3401990.
10. GNU General Public License, version 2. https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html. Cited April 3, 2026.
11. Source code of the libraries p4est and sc. https://github.com/cburstedde/p4est. Cited April 2, 2026.
12. GNU Lesser General Public License v2.1 - GNU Project - Free Software Foundation. https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html. Cited April 3, 2026.
13. G. M. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing: tech. rep.* (IBM, 1966).

Received	*Accepted*	*Published*
February 24, 2026	*March 23, 2026*	*April 17, 2026*

Information about the author

Andrey V. Solovjev — Ph. D., Leading Scientist; Lomonosov Moscow State University, Facility of Computational Mathematics and Cybernetics, Leninskie Gory, 1, building 52, 119991, Moscow, Russia.