

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Гетун Володимир Володимирович

**АНАЛІЗ ТА ОПТИМІЗАЦІЯ АЛГОРИТМІВ ПОБУДОВИ
ДИСКРЕТНОГО ПРЕДСТАВЛЕННЯ СКЛАДНИХ ОБ'ЄКТІВ**

**кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Мультимедійні системи»
за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис _____ Володимир ГЕТУН

Науковий керівник _____ Світлана ДОНЧЕНКО,
асистент кафедри інформаційних
технологій та систем

В.о. завідувача кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2024

АНОТАЦІЯ

Гетун В. В.

Тема: Аналіз та оптимізація алгоритмів побудови дискретного представлення складних об'єктів.

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ЛНУ імені Тараса Шевченка, 2024р.

Магістерська робота містить: 79 с., 26 рис., 4 табл., 46 джерел.

Об'єкт дослідження – алгоритми побудови дискретного представлення просторових конструкцій.

Предмет дослідження – оптимізація алгоритмів побудови дискретного представлення складних об'єктів.

Мета роботи – аналіз та оптимізація алгоритмів побудови дискретного представлення складних об'єктів та розробка додатку для тривимірної тріангуляції просторових конструкцій.

Результати роботи. Проаналізовано способи подання геометричних тіл та методи побудови сіток. Розглянуто та проаналізовано критерії якості побудованої сітки. Зроблено огляд сучасних програм для дискретизації просторових конструкцій.

Розроблено алгоритми побудови конформних тетраедральних сіток, узгоджених із заданою дискретною межею, для багатокomпонентних та багатогранних областей. Доведено кінцівку числа операцій і коректність роботи всього ланцюжка за умови використання точних обчислень. Доведено коректність роботи алгоритму рухомого фронту.

Розроблено додаток для тривимірної тріангуляції просторових конструкцій. У додатку доступно створення 3-х мірних сіток та їх оптимізація. Програма реалізується на мові програмування C++ в середовищі MS Visual C++ 2022. При створенні засобів відображення та візуалізації використовується графічний інтерфейс OpenGL.

Ключові слова: ТРІАНГУЛЯЦІЯ, АЛГОРИТМ, ТРИВИМІРНА ТРІАНГУЛЯЦІЯ, РУХОМИЙ ФРОНТ, ОПТИМІЗАЦІЇ СІТКИ, MS VISUAL C++, OPENGL.

ANNOTATION

Hetun Volodymyr

Theme: Analysis and optimization of algorithms for constructing a discrete representation of complex objects.

Speciality: 121 "Software Engineering".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2024 year.

Master's work of: 79 p., 26 im, 46 sources.

A research object of: - algorithms for building a discrete representation of spatial structures.

The article of research- optimization of algorithms for constructing a discrete representation of complex objects.

An aim of research is - analysis and optimization of algorithms for constructing a discrete representation of complex objects and development of an application for three-dimensional triangulation of spatial structures.

Job performanes - Methods of representation of geometric bodies and methods of construction of grids are analyzed. The criteria of quality of the constructed grid are considered and analyzed. An overview of modern programs for sampling spatial structures.

Algorithms for constructing conformal tetrahedral grids consistent with a given discrete boundary for multicomponent and multifaceted domains have been developed. The finite number of operations and the correctness of the whole chain are proved under the condition of using accurate calculations. The correctness of the algorithm of the moving front is proved.

An application for three-dimensional triangulation of spatial structures has been developed. The application provides the creation of 3-dimensional grids and their optimization. The program is implemented in the C ++ programming language in the environment of MS Visual C ++ 2022. When creating tools for display and visualization, the graphical interface OpenGL is used.

Keywords: TRIANGULATION, ALGORITHM, THREE-DIMENSIONAL TRIANGULATION, MOVING FRONT, GRID OPTIMIZATION, MS VISUAL C ++, OPENGL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО МОДЕЛЮВАННЯ І ДИСКРЕТИЗАЦІЇ ПРОСТОРОВИХ ОБ'ЄКТІВ.....	12
1.1. Аналіз відомих способів подання геометричних тіл.....	12
1.2. Аналіз методів побудови сіток	19
1.3. Оцінка якості сітки.....	25
1.4. Огляд існуючих програм для дискретизації.....	27
1.5. Висновки до розділу	32
РОЗДІЛ 2. ДОСЛІДЖЕННЯ АЛГОРИТМІВ ТРИВИМІРНОЇ ТРІАНГУЛЯЦІЇ ПРОСТОРОВИХ КОНСТРУКЦІЙ.....	33
2.1. Алгоритм рухомого фронту	34
2.1.1. Перевірка перетину тетраедра з трикутником	38
2.1.2. Скінченність роботи алгоритму рухомого фронту.....	39
2.2. Стійкий метод на основі тетраедризації Делоне.....	41
2.2.1. Тетраедризація Делоне	41
2.2.2. Відновлення геометрії області.....	42
2.2.3. Відновлення сліду сітки на кордоні	44
2.2.4. Скінченність роботи алгоритму.....	46
2.3. Поліпшення якості отриманої сітки	46
2.3.1. Алгоритми оптимізації сіток.....	47
2.4. Результати експериментів роботи алгоритмів	52
2.4.1. Швидкість роботи алгоритму рухомого фронту.....	52
2.5. Висновки до розділу	53
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ ТРИВИМІРНОЇ ТРІАНГУЛЯЦІЇ ПРОСТОРОВИХ КОНСТРУКЦІЙ ...	54
3.1. Об'єктно-орієнтована модель розрахункової сітки	54
3.2. Обґрунтування вибору середовища розробки програмного додатку	59
3.3. Розробка інтерфейсу	66

3.4. Генерація сітки в програмному додатку	72
3.5. Оптимізація сітки в програмному додатку	77
3.6. Формати файлів в програмному додатку	78
3.7. Висновки до розділу	80
ВИСНОВКИ	81
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	82
ДОДАТОК А.....	87

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СЕ	Скінчений елемент
САПР	Система автоматизованого проектування
ІМ	Інформаційна модель
ООП	Об'єктно - орієнтований підхід
ППП	Пакет прикладних програм

ВСТУП

Завдяки значному прогресу в області обчислювальної техніки чисельні методи стали основним інструментом математичного моделювання. При цьому з ряду причин найбільшого поширення набули проекційно - сіткові методи, зокрема, метод скінчених елементів. Їх використання передбачає попередню побудову так званої "сітки", тобто якоїсь топологічної безлічі точок ("вершин", "вузлів"), пов'язаних між собою "ребрами" - відрізками прямих (а в деяких випадках і кривих) ліній таким чином, що вихідна область розбивається на елементи певної форми. При цьому в якості елементів сітки, якщо мова йде про геометрично складні області, зазвичай використовуються геометричні симплекси, тобто трикутники в двовимірному і тетраедри в тривимірному випадку. Процес побудови сітки зазвичай називається дискретизацією або триангуляцією (навіть якщо мова йде про тривимірні випадки). При цьому до розмірів і форми елементів також пред'являються певні вимоги, так як вони істотно впливають на помилку апроксимації і збіжність методів. У рамках даної роботи розглядаються виключно симплексні елементи - тетраедри в тривимірному випадку, як найбільш універсальні і поширені.

Незважаючи на те, що дискретизація області зазвичай є лише одним з етапів застосування деякого чисельного методу, її часто виділяють в окрему задачу, так як, по-перше, дискретизація пов'язана з методом лише опосередковано, через вимоги до сітки, і, як правило, може бути проведена незалежно, а по-друге, рішення цього завдання вимагає використання зовсім іншого комплексу методів і алгоритмів, ніж ті, що застосовуються на інших етапах чисельного методу. Подібна відособленість дозволяє розглядати завдання дискретизації окремо, безвідносно якогось конкретного сіткового методу: на одній і тій же сітці, як правило, можна реалізувати цілий ряд самих різних методів, причому не тільки скінченно - елементних, а й різницевих. З цієї причини побудову розрахункової сітки часто називають однією з основних задач чисельних методів.

Розвиток обчислювальної техніки сприяв значному прогресу в області чисельних методів, зокрема, і в області методів тріангуляції. Розроблено нові класи методів, значно більш ресурсномісткі, але разом з тим і більш ефективні. Водночас під багато емпіричних методів тріангуляції підведена теоретична база.

В даний час двовимірна тріангуляція (без адаптації до рішення) є фактично закритою проблемою. Розроблені та теоретично обґрунтовані ефективні та надійні методи побудови й оптимізації сіток; властивості елементів - трикутників - добре вивчені. Разом з тим проблема тривимірної дискретизації ще далека від остаточного розв'язання: велика частина методів теоретично не обґрунтована, а багато завдань взагалі не вирішені.

На перший погляд подібна відмінність здається дивною, адже зазвичай математичні методи, розроблені для випадку двох вимірів, легко переносяться на випадок трьох і більше вимірів. На жаль, тривимірний простір має ряд особливостей, які ускладнюють подібний перенос.

Наприклад, площину можна елементарно заповнити правильними трикутниками; простір правильними тетраедрами заповнити не можна. Це основна перешкода на шляху створення якісних тривимірних сіток: оскільки в якості елементів неможливо використовувати правильні тетраедри, доводиться обходитися їх подобами, що негативно позначається на апроксимаційних властивостях сітки.

Більш того, будь який трикутник можна розбити на трикутники, подібні йому (на 4, 9, 16 і т.д.). Тетраedr в загальному випадку не можна розбити на подібні тетраедри. Це є основною перешкодою на шляху використання методів дроблення, які ефективно застосовуються в двовимірному випадку.

Будь який багатокутник на площині можна ребрами розбити на непересічні трикутники. У загальному випадку довільний багатогранник не можна розбити на непересічні тетраедри, не використовуючи додаткових вершин. Ця проблема може бути названа наріжною, оскільки істотно ускладнює використання практично всіх алгоритмів тріангуляції.

Крім зазначених складнощів теоретичного плану, є складнощі практичного характеру. Наприклад, неможливість (а точніше кажучи, вкрай висока складність) здійснення людського контролю над процесом тріангуляції. Якщо в плоскому випадку завжди можна вивести результат на дисплей для подальшої його корекції оператором, то для об'ємних областей це представляється вже вельми скрутним. Зважаючи на це, на методи тривимірної тріангуляції накладаються додаткові вимоги щодо надійності роботи і правильності побудови. Крім того, слід враховувати і значне збільшення споживаних ресурсів за більшого числа просторових вимірів (і, відповідно, кількості елементів сітки).

Проблема оптимальної дискретизації досліджуваної області на скінченні елементи в загальному вигляді є досить складною (особливо для тривимірних областей). Це обумовлено тим, що на форму скінчених елементів (СЕ) накладаються два основних обмеження: вони не повинні мати надто малих (або відповідно занадто великих) кутів і обсяг СЕ не повинен перевищувати деяку задану величину. У першому випадку при розрахунках виникають значні обчислювальні похибки. У другому з'являється ризик втрати точності обчислень при значній зміні градієнта досліджуваної функції.

Можливостей стандартних генераторів сіток часто не вистачає для побудови сіток із заданими характеристиками, що обумовлює необхідність розробки власних програм, які можливо адаптувати до розв'язуваної проблеми.

Тому автоматична генерація СЕ-сітки являє собою досить складну процедуру, яка є основою будь-якого скінчено-елементного пакету програм і являється актуальною.

Об'єкт дослідження – алгоритми побудови дискретного представлення просторових конструкцій.

Предмет дослідження – оптимізація алгоритмів побудови дискретного представлення складних об'єктів.

Мета роботи – аналіз та оптимізація алгоритмів побудови дискретного представлення складних об'єктів та розробка додатку для тривимірної тріангуляції просторових конструкцій.

Методи дослідження: методи обчислювальної математики і комп'ютерної графіки.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати основні поширені способи подання геометричних тіл та методи й алгоритми побудови сіток просторових областей;
- розглянути та проаналізувати критерії якості побудованої сітки;
- розробити алгоритми побудови конформних тетраедральних сіток, узгоджених із заданою дискретною межею, для багатокомпонентних та багатограних областей;
- розробити і реалізувати додаток для тривимірної тріангуляції просторових конструкцій.

Практичною цінністю роботи є розробка додатку автоматичної генерації тривимірних розрахункових сіток для скінчено – елементного моделювання конструкцій.

В першому розділі в результаті огляду літературних джерел проаналізовано способи подання геометричних тіл та методи побудови сіток. Розглянуто та проаналізовано критерії якості побудованої сітки. Зроблено огляд сучасних програм для дискретизації просторових конструкцій.

В другому розділі розроблено алгоритми побудови конформних тетраедральних сіток, узгоджених із заданою дискретною межею, для багатокомпонентних та багатограних областей. Доведено кінцівку числа операцій і коректність роботи всього ланцюжка за умови використання точних обчислень. Доведено коректність роботи алгоритму рухомого фронту.

У третьому розділі аргументується вибір середовища розробки додатку і мови програмування. Описано алгоритм генерації сіток. У додатку доступно створення 3-х мірних сіток і їх оптимізація. Програма реалізується на мові

програмування C++ в середовищі Visual C++ 2022. При створенні засобів відображення та візуалізації використовується графічний інтерфейс OpenGL.

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО МОДЕЛЮВАННЯ І ДИСКРЕТИЗАЦІЇ ПРОСТОРОВИХ ОБ'ЄКТІВ

1.1. Аналіз відомих способів подання геометричних тіл

Загальна класифікація основних підходів до моделювання геометричних тіл була дана в роботі [21], там же були сформульовані і перераховані нижче основні вимоги до твердотілих моделей:

- показність - модель повинна бути придатна для опису безлічі фізичних об'єктів досить широкого класу;
- однозначність - один і тільки один об'єкт повинен відповідати кожному конкретному опису, щоб не виникало питання про те, який власне об'єкт представляється;
- унікальність - в ідеалі бажано, щоб кожен модельований об'єкт описувався в обраній схемі подання єдиним чином. Ця властивість забезпечує легкість розрізнення двох різних об'єктів, але на практиці є важко досяжною;
- точність - бажано, щоб подання точно описувало форму об'єкта, без апроксимації;
- коректність - в ідеалі, схема подання повинна допускати введення тільки тих описів, які задовольняють всім критеріям визначення твердотілих моделей;
- замкнутість - подання має бути замкнутим щодо допустимих в ньому операцій. Для геометричного моделювання велике значення мають геометричні перетворення та теоретико - множинні операції;
- компактність - модель повинна мати компактний опис, економне з точки зору даних, необхідних для її повного завдання;
- ефективність - модель повинна допускати ефективні алгоритми її обробки, що охоплює введення/висновок, обчислення основних відносин і операцій, редагування і модифікацію, візуалізацію, обчислення метричних характеристик і т.д.

Задовольнити всім цим вимогам одночасно важко, тому різні схеми подання будуються на основі деякого компромісу і мають свої переваги і недоліки.

В даний час можна виділити наступні найбільш поширені методи представлення геометричних тіл [4, 21, 22, 23]:

- параметризовані примітиви;
- граничне уявлення;
- конструктивна геометрія;
- кінематичний метод ("свіппінг" або замітання);
- розкладання на елементи;
- просторове перерахування;
- неявні моделі.

Моделювання за допомогою бібліотеки параметризованих примітивів застосовується зазвичай в різних прикладних областях, де набір використовуваних геометричних об'єктів обмежений і стандартизований. Такий підхід відповідає груповій технології, застосовуваний в автоматизованому проектуванні. Він часто використовується для таких складних і в той же час стандартизованих об'єктів, як болти, зубчасті шестерні і т.п., які втомливо визначати за допомогою булевих комбінацій більш простих об'єктів, але можна охарактеризувати набором високорівневих параметрів (діаметр і кількість зубів для шестерні і так далі).

При використанні граничного опису геометричне тіло задається замкнутою поверхнею, що обмежує це тіло. При цьому для опису форми області використовують як алгебраїчні поверхні 1 -го і 2 -го порядку, так і кусково-аналітичні поверхні. За допомогою граничного опису може бути представлений широкий клас об'єктів. У граничному поданні в явному вигляді міститься інформація про поверхні тіла, дане подання ефективно при візуалізації та чисельному моделюванні. Основним недоліком методу є громіздкість даних, що описують модель і складність обчислення теоретико-множинних операцій.

У методі конструктивної геометрії складний об'єкт формується шляхом виконання теоретико-множинних операцій та операцій геометричних перетворень над простішими об'єктами, званими базовими елементами, або примітивами. Для того, щоб результат застосування теоретико-множинних операцій до твердотілих примітивів сам був твердим тілом, виконується його регуляризація, яка зводиться до замикання внутрішніх точок. Таким чином визначаються регуляризовані теоретико-множинні операції [4].

Модель конструктивної геометрії може бути описана за допомогою дерева побудови [24], в якому нетермінальні вузли представляють оператори, а листя - базові елементи. Метод конструктивної геометрії охоплює широке коло об'єктів, його зручно використовувати при введенні, тому що в ньому інформація про об'єкт представлена в досить структурованій формі, що забезпечує велику наочність і дозволяє уникнути помилок при описі об'єкта. Крім того, подання за допомогою дерева побудови ефективно при редагуванні. Існують алгоритми візуалізації тіл, представлених безпосередньо за допомогою методу конструктивної геометрії, проте часто на етапі отримання зображення здійснюють перехід до граничного опису.

У кінематичному методі [25] двовимірна область представляється як слід рухомої у просторі кривої, а тривимірна область - як слід рухомого двовимірного тіла або перетину. Подальшим розвитком кінематичного методу є, так званий, плазовий метод, в якому об'єкт, що рухається по складній траєкторії, не є жорстким, а деформується відповідно до залежностей тій чи іншій мірі складності. Кінематичний метод зручний при введенні, дозволяє ефективно виконувати ряд обчислювальних операцій, проте він охоплює обмежений клас об'єктів і не передбачає виконання теоретико-множинних операцій.

У методі розкладання на елементи модельований об'єкт представляється як об'єднання деякого набору неперекриваючихся елементів - примітивів. Цей метод близький до методу конструктивної геометрії, проте у ньому, в

порівнянні з останнім, звужене коло перетворень, виконуваних над примітивами в процесі завдання об'єкта.

Дане обмеження значно ускладнює формування опису об'єкта, однак при цьому спрощується виконання операції розбиття, що ефективно для візуалізації та чисельного моделювання.

У методі просторового перерахування об'єкти задаються шляхом перерахування всіх тих позицій простору, які вони займають. При цьому вважають, що простір складено з елементарних осередків, що примикають один до одного, а об'єкт представлений як об'єднання деякої кінцевої безлічі таких осередків. В якості осередків використовують зазвичай замкнуті квадрати (куби) фіксованого розміру зі сторонами, паралельними координатним осям (площинам). Відомі дві разновидности методів просторового перечислення - воксельні представлення і представлення за допомогою квадратичних (вісімкових) дерев, які відрізняються способом описання сукупності осередків, які займає модельований об'єкт. У першому із зазначених методів для цих цілей використовуються матричні структури, а в другому - ієрархічні структури - квадратичні і восьмиричні дерева [26]. Розроблено спеціальні дуже ефективні алгоритми для виконання теоретико-множинних операцій і візуалізації воксельних моделей і моделей на основі квадратичних (вісімкових) дерев.

У неявних моделях тіло задається за допомогою деякої процедури, яка дозволяє для кожної точки простору моделювання визначати її приналежність описуваному об'єкту [27, 28]. Ця процедура може бути реалізована різними способами. Наприклад, у вузлах регулярної сітки, що охоплює цілком модельований об'єкт, можна задати предикат приналежності й обчислювати приналежність інших точок за допомогою інтерполяції [29]. Для неявного опису геометричних тіл використовують також функцію відстані, значення якої в кожній точці дорівнює відстані від цієї точки до модельованого об'єкта [28]. Така функція може бути також визначена на базі воксельного подання. Нарешті, існує підхід, який в загальному вигляді неявно описує довільний

геометричний об'єкт в просторі E_n за допомогою функції $f(x_1, x_2, \dots, x_n)$ координат точок (x_1, x_2, \dots, x_n) у вигляді нерівності $f(x_1, x_2, \dots, x_n) \geq 0$, так що функція $f()$, що визначає об'єкт, приймає позитивні значення в точках, що лежать в середині об'єкта, нульове - в граничних точках і негативне - в точках поза об'єктом. Такий підхід отримав назву функціонального подання (F-per) [2]. Як і в методі конструктивної геометрії, складний F-per об'єкт, може формуватися з простих за допомогою теоретико-множинних операцій і геометричних перетворень.

Аналітичний вид функції, що описує результат теоретико-множинних операцій, може бути знайдений на основі теорії R-функцій [30]. Застосування R-функцій дозволяє отримувати опис поверхні складного геометричного тіла в неявному вигляді $f(x, y, z) = 0$. В даний час розроблені методи параметризації таких поверхонь [31, 32], що забезпечують побудову графічних відображень тіл, представлених за допомогою функціональних моделей. Функціональне уявлення узагальнює різні способи неявного завдання геометричних тіл, в рамках нього реалізуються різноманітні операції, в тому числі замітання, декартовий твір, метаморфозіс та ін.

Аналізуючи описані уявлення з точки зору зазначених вище вимог, що пред'являються до твердотілих моделей, можна відзначити наступне. Серед перерахованих уявлень низьку точність мають методи просторового перерахування, точність граничних уявлень, конструктивних моделей і розкладання на елементи залежить від форми сегментів поверхонь і об'ємних примітивів. При цьому найбільш широке коло об'єктів може бути представлене методами просторового перерахування, граничним поданням, функціональним і конструктивним методами. Методи замітання і розкладання на елементи, а також представлення екземплярами примітивів вельми обмежені в сенсі безлічі представимих тіл. Унікальність уявлення практично гарантують тільки воксельні моделі та вісімкові дерева при додатковому підрозбитті. Серед всіх уявлень, найбільш важко оцінити коректність для граничного подання, де не тільки структури даних, що представляють

вершини, ребра і грані можуть бути суперечливими, але також грані або ребра можуть перетинатися. Складність аналізу модельованого об'єкта на предмет відповідності визначенню твердого тіла характерна і для функціонального подання. Більш легко перевірити коректність для конструктивної геометрії і розкладання на елементи. Найбільш просто оцінити коректність моделей просторового перерахування. Регуляризовані теоретико - множинні операції не визначені для подання екземплярами примітивів, кінематичних моделей і розкладання на елементи. Решта моделей замкнуті щодо цих операцій.

Найбільш компактний опис мають уявлення на базі параметричних примітивів, функціональні, конструктивні та кінематичні моделі. Описи моделей просторового перерахування громіздкі, але мають просту структуру. Найбільш складно і громіздко описуються граничні подання. Ефективність реалізації операцій сильно відрізняються у різних геометричних моделей.

Візуалізація та чисельні розрахунки найбільш просто реалізуються для моделей просторового перерахування і розкладання на елементи.

Полігонізація поверхні, необхідна для відображення 3D об'єктів, найбільш просто будується для граничних моделей і кінематичних моделей. Обчислення межі є трудомісткою операцією для конструктивної геометрії і особливо для функціонального подання.

Що ж до регуляризованих булевих операцій, то вони природним чином виконуються для конструктивної і функціональної моделей, порівняно легко обчислюються для воксельних моделей і моделей на основі вісімкових дерев і значно більш складно реалізуються для граничних моделей. Питання про приналежність точки модельованого об'єкту найбільш просто вирішується для функціонального уявлення і методу просторового перерахування. Набагато складніше це завдання вирішується для інших моделей.

Зазначена класифікація відображає принципові підходи до моделювання тіл. В рамках кожного з перерахованих типів існує безліч конкретних моделей, що відрізняються способом реалізації. Жоден із зазначених способів не може вважатися повністю універсальним, ефективність тієї чи іншої моделі

залежить від операцій, які необхідно виконувати над об'єктами в процесі моделювання. Тому вибір моделі залежить від прикладної задачі. Перераховані методи опису геометричних тіл доповнюють один одного, тому найбільш ефективним часто виявляється спільне використання декількох уявлень, що припускає їх взаємні перетворення. Проте реалізації моделей різних типів сильно відрізняються по використовуваному математичному апарату, структурам даних і алгоритмам обробки.

При вирішенні багатьох прикладних задач виникає необхідність розбиття модельованих об'єктів і побудови дискретних моделей, які з заданою точністю апроксимують форму вихідного об'єкта. У дискретному поданні важливу роль відіграють засоби опису внутрішньої структури модельованих об'єктів. Для опису дискретних моделей використовуються різні стратифікації, їх огляд подано в роботі [33].

Загалом стратифікація - це опис об'єкта у вигляді об'єднання сукупності непересічних підмножин, кожна підмножина є різноманіттям у просторі E_n , межа кожної зв'язної компоненти підмножини має розмірність нижче, ніж розмірність самої підмножини і в кожній обмеженій множині простору моделювання міститься кінцеве число елементів [34]. Окремими випадками стратифікацій є геометричні комплекси [35, 36, 37].

У рамках наведеної вище класифікації дискретні моделі можна віднести до типу розкладання на елементи. Опис просторової структури також важливий при завданні складових кривих і поверхонь, кусочно-аналітичному поданні граничних моделей і при завданні розмірно неоднорідних об'єктів. Для цих цілей також застосовуються різні стратифікації [39, 40]. Зокрема, в роботі [40] було показано, що геометричні комплекси дозволяють єдиним чином представляти граничні моделі, кінематичні моделі та моделі просторового перерахування. Відзначимо також, що в залежності від способу опису примітивів в конструктивному представленні його можна звести відповідно до граничної моделі, моделі просторового перерахування або функціональної моделі. Таким чином, в якості основної альтернативи у виборі

уявлення геометричних тіл можна розглядати модель на основі комплексів, звану також клітинною, і функціональне уявлення. Перша з цих моделей задає явний опис об'єкта, а друга - неявний. Ці моделі принципово відрізняються, однак вони не замінюють один одного, кожна з них має свої переваги і недоліки. Так функціональне уявлення забезпечує компактний опис складної, можливо багатовимірної, геометрії, воно придатне для опису об'єктів різної розмірності, включаючи розмірно неоднорідні об'єкти та об'єкти, які не є різноманітні. Однак воно не містить даних про топологічну структуру об'єкта, що викликає проблеми при виконанні багатьох чисельних процедур і операцій. У свою чергу клітинне уявлення, засноване на топологічному розбитті, дає повний опис топологічної структури об'єкта, дозволяє виділяти топологічно однорідні компоненти в складі складних об'єктів. Однак ступінь деталізації клітинного уявлення, що визначається топологічним розбиттям, часто виявляється надлишковою з точки зору опису геометричних властивостей. У додатках, які не використовують повною мірою інформацію про топологічну структуру об'єкта, така надмірна деталізація знижує ефективність роботи з геометричним об'єктом.

Враховуючи вище викладене, можна зробити висновок, що при моделюванні складних неоднорідних об'єктів різні уявлення можуть вдало доповнювати один одного. На практиці для забезпечення можливості спільного використання різних уявлень необхідно забезпечити узгодженість специфікацій різних моделей з урахуванням їх взаємних перетворень.

1.2. Аналіз методів побудови сіток

Процес побудови сіток може розглядатися як перетворення геометричних моделей. Методи дискретизації істотно залежать від способу подання об'єкту, що розбивається, і від обмежень, накладених прикладним завданням на дискретну модель.

Огляд алгоритмів побудови сіток можна знайти зокрема в роботах [40, 41]. Серед різних способів генерації сіток можна виділити наступні групи найбільш поширених методів:

- методи тетраедризації (тріангуляції);
- методи побудови неструктурованих гексагональних (чотирикутних) сіток;
- кінематичні методи або методи об'єднання перетинів;
- методи накладення неузгоджених з формою області сіток;
- методи відображень геометрично нерегулярних областей в області правильної форми;
- методи оптимізації.

Аналіз різних методів тріангуляції і тетраедризації дається в роботах [42, 43]. Серед методів тріангуляції можна виділити дві основні групи. У методах першої групи вихідною інформацією для генерації сіток є набір вузлів, які потім з'єднуються, утворюючи трикутні елементи або тетраедри. Найбільш часто при цьому використовується метод Делоне, в якому вузли з'єднуються таким чином, що всередину кола (кулі), описаного навколо формованого трикутника (тетраедра), не потрапляють ніякі інші вузли сітки, відмінні від вершин даного елемента.

Існують різні способи початкового вибору вузлів. Одні автори пропонують спочатку розміщувати вузли на межі області, а потім генерувати внутрішні вузли випадковим чином, залишаючи тільки ті з них, які задовольняють заданій щільності сітки. У ряді робіт спочатку вводиться набір горизонтальних прямих, що покриває всю область рішення. Потім вузли рівномірно розподіляються на відрізках цих прямих, що лежать всередині області. У тривимірному випадку для автоматичного завдання вузлів пропонується також використовувати представлення області з допомогою вісімкових дерев.

До другої групи методів тріангуляції відносяться так звані методи декомпозиції. У методах декомпозиції можна виділити рекурсивні та ітераційні методи.

У рекурсивних методах вихідна область спочатку розбивається на підобласті, які потім діляться в відповідності з тим же самим алгоритмом

розбиття. Процес продовжується до тих пір, поки розміри отриманих підобластей не відповідатимуть необхідній щільності сітки. Цей метод дозволяє будувати сітки в областях, які є багатокутниками, які в загальному випадку можуть бути і криволінійними. Вихідна груба сітка виходить шляхом побудови діагоналей, що з'єднують несуміжні вершини багатокутника. Елементи грубої сітки потім подрібнюються, для цього вводяться додаткові вузли, що розміщуються на границях елементів, ці вузли з'єднуються один з одним відрізками, що лежать всередині елементів. Даний метод допускає узагальнення й на тривимірний випадок.

В ітераційних методах декомпозиції розбиття здійснюється таким чином, що на кожному кроці будується один або кілька елементів результуючої сітки. Процес продовжується до тих пір, поки елементи сітки не заповнять всю розрахункову область. Методи цієї групи дозволяють будувати, як правило, трикутні та тетраедральні сітки. В ітераційних методах існує можливість управління розмірами і формою елементів безпосередньо в процесі побудови сітки, що є їх безперечною перевагою порівняно з рекурсивними методами.

Ітераційні методи універсальні і, як правило, застосовуються для областей досить довільної форми. Саме тому ітераційні методи в основному і використовуються в автоматичних програмних комплексах. Недоліком цього класу методів є ресурсомісткість, істотно більш повільна швидкість роботи (у порівнянні з прямими методами) і менша надійність.

Ітераційні методи через свою універсальність отримали найбільший розвиток. Розроблено кілька різних підходів, які можна розділити на три підкласи: методи граничної корекції, методи на основі критерію Делоне і методи вичерпання.

Методи граничної корекції є найшвидшими з ітераційних методів, але, на жаль, мають ряд невикорінних недоліків. Побудова сіток в цих методах здійснюється в два етапи. На першому етапі проводиться триангуляція якоїсь простої "супер - області", яка повністю включає в себе задану область. Як

правило, ця супер - область являє собою паралелепіпед (через простоту тріангуляції), тріангуляція якого здійснюється на основі одного з численних шаблонів. На другому етапі всі вузли отриманої сітки, що лежать поблизу межі заданої області, проектується на поверхню межі; а вузли, що лежать поза заданою областю - видаляються. Щоб компенсувати неминучі геометричні спотворення елементів сітки поблизу межі, часто додатково проводять ще один етап - етап оптимізації сітки, що в підсумку дозволяє отримати досить хороші результати.

Очевидно, що даний метод не можна застосовувати для дискретизації областей із заданою тріангуляцією меж. Це істотне обмеження, а також інші складності знижують популярність методу, зводячи нанівець його основну перевагу - високу швидкість роботи.

Сутність методів вичерпання полягає в послідовному "вирізанні" із заданої області фрагментів тетраедричної форми доти, поки вся область не опиниться "вичерпана". В англomовній літературі цей метод отримав назву "advancing front", що також добре відображає ідею методу. Вихідними даними на кожній ітерації є "фронт", тобто тріангуляція межі ще не "вичерпана" частина області. Кожен трикутник цієї тріангуляції є основою області тетраедра, який вилучається, причому на кожній ітерації може вилучатися або один тетраедр, або відразу цілий шар тетраедрів. Після вилучення тетраедра "фронт" оновлюється, після чого відбувається перехід до наступної ітерації.

Методи вичерпання універсальні і можуть бути використані для областей довільної форми та конфігурації (навіть для незв'язних областей), що пояснює їх популярність. Зокрема, саме ці методи використовуються в програмному комплексі ANSYS. Разом з тим слід відзначити їх високу ресурсомісткість і низьку швидкість роботи.

Методи на основі критерію Делоне часто називають просто методами Делоне, хоча це не зовсім коректно, оскільки сам Б.Н. Делоне ніяких методів не розробляв, а лише запропонував простий і ефективний критерій, що використовується при установці зв'язків між вузлами. Відповідно, ідеєю цього

класу методів є розміщення в заданій області вузлів і подальша розстановка між ними зв'язків згідно з критерієм Делоне (чи іншому схожому критерію).

Неструктуровані гексагональні сітки будуються зазвичай на основі симпліціальних сіток шляхом об'єднання сусідніх елементів [40].

Кінематичний метод дозволяє будувати сітки в областях, заданих методом замітання [45]. При дискретизації області спочатку розбивається рухома крива (двовимірний розтин), а потім відповідні одна одній точки (відрізки) на сусідніх за часом кривих (перетинах) з'єднуються один з одним, утворюючи елементи двовимірної (тривимірної) сітки. Кінематичний метод є окремим випадком загального методу об'єднання перерізів. У методі об'єднання перерізів передбачається відома довільна послідовність отриманих яким-небудь способом перерізів об'єкта, який розбивається. Тоді, якщо на перетинах вдається побудувати ізоморфні сітки, то з'єднуючи відповідні один одному елементи перерізів, можна отримати чотирикутну або призматичну сітку, яка є дискретизацією розглянутого об'єкта. Даний спосіб є досить ефективним, однак він застосовується до обмеженого кола об'єктів. У деяких випадках, коли сітки на сусідніх перерізах не є ізоморфними, для дискретизації простору між такими перерізами використовуються методи тріангуляції.

При використанні неузгоджених сіток вихідна область покривається деякої регулярною сіткою. Сіткові елементи всередині області залишаються без змін, а елементи, що лежать на межі і поза областю ігноруються. У результаті виходить сітка зі ступінчастою або ламаною межею, для згладжування якої можуть застосовуватися різні підходи. Зокрема, проєціювання точок ступінчастої межі на межу області, або зсув приграничних вузлів уздовж ліній сітки до перетину з межею.

Ефективний підхід до побудови неузгоджених сіток пов'язаний з використанням представлення області з допомогою вісімкових дерев. Цей метод дозволяє в автоматичному режимі здійснювати дискретизацію складних тривимірних об'єктів [44].

Для генерації сіток активно застосовуються також різні відображення [45, 46], що дозволяють перетворювати сітки, поставлені на областях правильної форми, сітки, що покривають області складної форми. В даний час найбільше поширення одержали конформні відображення і трансфінитні перетворення. Методи, засновані на використанні різних відображень, що забезпечують побудову, як правило, якісних розрахункових сіток. Однак застосування подібних алгоритмів пов'язане з істотними обмеженнями на форму вихідної області і на спосіб подання цієї області, що призводить до необхідності індивідуального підходу до вирішення кожної нової задачі.

Методи відображень дозволяють будувати блочно-структуровані сітки в галузях, визначених методом розкладання на елементи, що припускає завдання галузі у вигляді об'єднання макроблоків певної форми. Для автоматичного розбиття складних областей на макроблоки застосовується також перетворення, яке дозволяє виділити кістяк об'єкта, що складається з безлічі точок, кожна з яких є рівно віддаленою від найближчих до неї точок межі об'єкта [30].

Методи оптимізації сіток використовуються для покращення та адаптації до умов поставленої задачі вже сформованих яким-небудь способом сіток [31, 32, 33]. У загальному вигляді суть цих методів можна сформулювати наступним чином:

$$\text{необхідно знайти } E = \min \left(\max_i E_i \right), i=1,2,\dots,N,$$

де E_i - якась міра помилки, що дозволяє кількісно оцінити якість кінцево-різницевої або скінчено-елементної апроксимації - E_i -помилка на i -му елементі, N - загальна кількість елементів. Міра E може бути обрана самими різними способами, вона може залежати тільки від геометричних характеристик елементів або ж враховувати характер прикладної задачі. При оптимізації сіток або варіюють тільки координатами вузлів, залишаючи топологію без зміни, або допускають зміну топології, в цьому випадку можливе розбиття або навпаки укрупнення деяких елементів, зміна

конфігурації зв'язків між вузлами або введення додаткових вузлів на межах і всередині елементів.

Кожен з перерахованих методів побудови сіток розрахований на певну модель подання розрахункової області [34]. Так при тріангуляції, заснованої на методах декомпозиції, використовується граничний опис області. При побудові неузгоджених сіток застосовується завдання області методом просторового перерахування або граничним методом. Кінематичний метод придатний для розбиття тільки тих областей, опис яких також будується на основі кінематичного методу. Методи відображень припускають завдання розрахункових областей або методом розкладання на елементи, або граничним методом.

Слід зазначити, що застосування тих або інших способів побудови сіток істотний вплив мають обмеження, що накладаються методами, використовуваними для вирішення прикладних завдань.

Таким чином, вибір типу розрахункових сіток і алгоритмів їх побудови з одного боку залежить від прикладної задачі і методів її чисельного рішення, а з іншого боку визначається способом завдання геометричної моделі розрахункової області.

1.3. Оцінка якості сітки

Порівняння ефективності різних методів дискретизації неможливе без позначення якогось критерію якості побудованої сітки. Оскільки сітка будується не заради самої побудови, а для вирішення деякої задачі, розумно пов'язати цей критерій з апроксимаційними властивостями сітки. Відповідно до теорії, ці властивості в основному залежать від форми елементів [4]. Зокрема, в оцінку похибки апроксимації кінцевим елементом, як правило, входить величина

$$R(\omega)/diam(\omega) \quad (1.1),$$

де $R(\omega)$ - радіус вписаного в ω кулі, а $diam(\omega)$ - діаметр ω . Оскільки пряме знаходження (1.1) хоча і можливе, але занадто складне, на практиці використовуються різні альтернативні оцінки. Таких оцінок запропоновано велику кількість, деякі найбільш популярні наведені у таблиці 1.1. Однак

оптимальною з точки зору точності, повноти оцінки якості сітки і зручності знаходження є наступна оцінка [1, 31]:

$$\mu = \frac{V}{abc} \quad (1.2),$$

де V - об'єм тетраедра, а abc - найбільше з добутків довжин трійки ребер, що виходять з однієї вершини. Далі ми будемо використовувати саме цю оцінку.

Оскільки величина (1.2) має порядок десятих і сотих, для наочності її зручно відносити до значення ідеального випадку - правильного тетраедра. (Для нього, як можна підрахувати, ця величина дорівнює $\sqrt{2}/12 \approx 0.118$). Назвемо це відношення (μ з (1.2) до ідеального значення) "апроксимаційною характеристикою" (АХ) елемента. Можливі значення АХ лежать в межах від 0 до 1; чим ближче до 1, тим краще.

Для якісного аналізу сітки найбільшу важливість мають мінімальне та середнє значення АХ: перше бере участь в оцінках якості апроксимації, другий свідчить про загальну якість сітки. Для геометрично складних областей хорошим результатом буде середнє значення АХ, рівне хоча б $1/2$.

Таблиця 1.1

Критерії оцінки якості елементів сітки

Критерій	Формула	Інтервал можливих значень	Оптимальне значення
Ставлення радіусу описаної сфери до радіусу вписаної	$\beta = \frac{R_c}{R_i}$	$[1, +\infty)$	3.0
Відношення довжини найбільшого ребра до радіусу вписаної окружності	$\sigma = \frac{L_{\max}}{R_i}$	$[1, +\infty)$	4.898979...
Відношення радіусу описаного кола до довжини найбільшого ребра	$\omega = \frac{R_c}{L_{\max}}$		0.612375...
Відношення довжин найбільшого і найменшого ребер	$\tau = \frac{L_{\max}}{L_{\min}}$	$[1, +\infty)$	1.0

Відношення 4-го ступеня об'єму тетраедра до кубу суми квадратів площ граней	$k = \frac{V^4}{(\sum_{i=0}^3 S_i^2)^3}$	$(0,1]$	4.572474e-4
Відношення куба середнього арифметичного довжин ребер до обсягу тетраедра	$\alpha = \frac{\bar{L}^3}{V}$	$[1,+\infty)$	8.4852816...
Відношення куба середнього геометричного довжин ребер до обсягу тетраедра	$\gamma = \frac{\bar{L}^3}{V}$	$[1,+\infty)$	8.4852816...
Найбільший двогранний кут	δ	$\left[\arccos \frac{1}{3}, \pi \right)$	$\arccos \frac{1}{3}$ (1.2309594...)
Мінімальний тілесний кут	θ	$(0, \pi / 2]$	$\pi / 2$
Відношення об'єму тетраедра до найбільшого з добутків довжин трійки ребер, які виходять з однієї вершини	$\eta = \frac{V}{\max_i \left(\prod_{\substack{j=1 \\ i \neq j}}^4 l_{ij} \right)}$	$(0, \sqrt{2} / 12]$	$\sqrt{2} / 12$

1.4. Огляд існуючих програм для дискретизації

Сітка робить досить істотний вплив на точність рішення проєкційно - сітковими методами (а в деяких випадках і на збіжність методів). Якість сітки з точки зору точності рішення визначається трьома обставинами: розмірами елементів, їх формою і тим, наскільки добре сітка апроксимує вихідну область. Таким чином, можна сформулювати такі вимоги до будь-якої системи автоматичної триангуляції (САТ).

1. Максимально точна апроксимація меж області та її внутрішніх обмежень: лінії з'єднань поверхонь повинні бути апроксимовані ланцюжками ребер сітки, а самі поверхні - безліччю плоских трикутних граней.
2. Форма елементів повинна бути по можливості близька до форми правильного симплекса.
3. САТ повинна дозволяти контролювати розміри елементів сітки, щоб можна було забезпечити згущення сітки в потрібних областях.

Важливою також є можливість перевірки якості та коректності побудованої сітки.

Нижче наводиться огляд різних програмних пакетів, призначених для дискретизації складних просторових областей (вибірка обмежена програмами з відкритим кодом, повністю безкоштовними програмами та програмами, безкоштовними для академічного використання).

Пакет Geompack розроблений доктором Баррі Джо, відомим своїми роботами в області дискретизації методами на основі критерію Делоне. Перші версії пакету з'явилися більше 20 років тому і мали - вид програмної бібліотеки на мові FORTRAN77 (старі версії пакету поширюються з відкритим кодом). Тепер Geompack є потужною консольною програмою для виконання різних операцій з сітками - побудови, згущення, оптимізації, перебудови і т.п. В основі алгоритму використовується метод корекції спільно з методом на основі критерію Делоне.

Пакет Geompack безкоштовний для академічного використання. У комплекті з пакетом поставляється докладна документація англійською мовою і безліч прикладів. Автор здійснює обмежену підтримку користувачів по електронній пошті. Візуальний інтерфейс і можливості по візуалізації відсутні. Імпорт та експорт даних здійснюється через текстові та бінарні файли спеціальних (нестандартних) форматів. Пакет може бути використаний під операційними системами сімейств Unix і Windows.

До недоліків пакета слід віднести неможливість прямого контролю над розмірами елементів.

Домашня сторінка пакета в Інтернеті: <http://members.shaw.ca/bjoe/>

TetGen - програма, розроблена доктором Хан Сі з інституту Вейерштраса (WIAS). В основі алгоритму лежить метод на основі критерію Делоне. Для обліку внутрішніх обмежень використовується метод перебудови. TetGen володіє візуальним інтерфейсом (може запускатися і як консольна програма), дозволяє будувати та оптимізувати сітки, оцінювати їх якість, а також виробляти локальне згущення.

Пакет TetGen безкоштовний для академічного використання. У комплекті з пакетом поставляється докладна документація англійською мовою

і безліч прикладів. Імпорт та експорт даних здійснюються через текстові файли різних (у тому числі і стандартних) форматів. Розміри елементів контролюються глобальним чином.

Пакет може бути використаний під будь-якими операційними системами, для яких існує компілятор C ++.

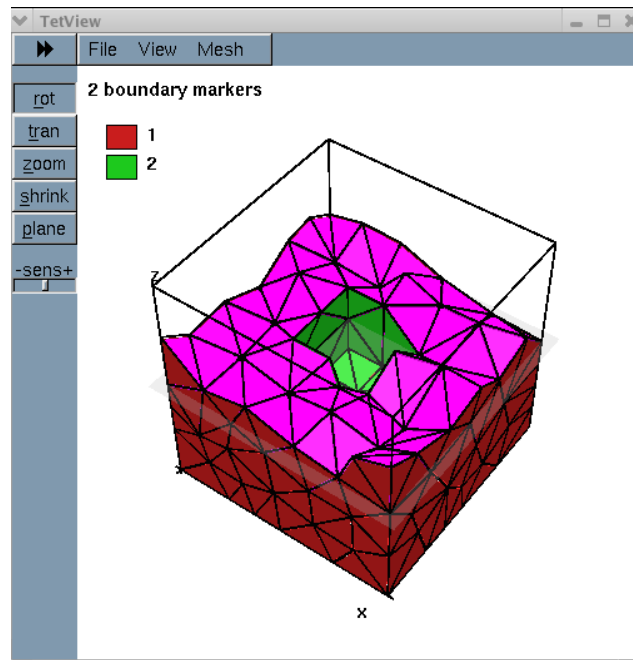


Рис. 1.2. Вікно програми TetGen

Домашня сторінка пакета в Інтернеті: <http://tetgen.berlios.de/>

Програмний комплекс "Mefisto", розроблений колективом французьких авторів під керівництвом Алана Перрона, призначений для вирішення різних завдань математичної фізики, і включає в себе модуль дискретизації, постпроцесингу, вирішувача і візуалізації. Модуль для дискретизації областей (Mefisto - maillages) можна використовувати і окремо.

Програмний комплекс Mefisto призначений для роботи під управлінням операційних систем сімейства Unix. Поширюється разом з вихідними кодами (написаний на мовах FORTRAN77 і C) і демонстраційними прикладами. Документація французькою мовою.

Для тріангуляції поверхні і внутрішній області використовується комбінований метод на основі критерію Делоне та алгоритму Quad - tree. Управління розмірами елементів здійснюється за допомогою спеціальних функцій (тобто згущення сітки можна отримати в процесі побудови). Область

для тріангуляції можна задавати за допомогою текстових файлів спеціального формату або візуального інтерфейсу. Експорт сітки проводиться в текстові файли спеціального формату.

Домашня сторінка програмного комплексу в Інтернеті:
<http://www.ann.jussieu.fr/~perronnet/mefistoa.gene.html>

NetGen - один з найпопулярніших безкоштовних пакетів для дискретизації складних областей. Поширюється з відкритим кодом, супроводжується документацією англійською мовою, безліччю прикладів, а також має власний Інтернет -форум. Може бути використаний під будь-якими операційними системами, для яких існує компілятор C++. Володіє візуальним інтерфейсом, але може бути запущений і як консольна програма. Дозволяє згущувати та оптимізувати сітки.

NetGen призначений для генерації і відображення 3D- сіток і складається з декількох бібліотек. Основна бібліотека `nglib` реалізує безпосередньо генерацію сіток. Є два способи завдання простору, яке має бути заповнене тетраедальними елементами:

Завдання простору за допомогою операторів конструктивної блокової геометрії (Constructive Solid Geometr, CSG). У цій технології шуканий простір описується сукупністю примітивних об'єктів, над якими задаються такі операції як об'єднання, перетин, різниця.

Опис поверхні, що є границею шуканого простору, у файлі формату STL. У цьому форматі інформація про поверхні представляється у вигляді координат трикутних граней поверхні і їх нормалей.

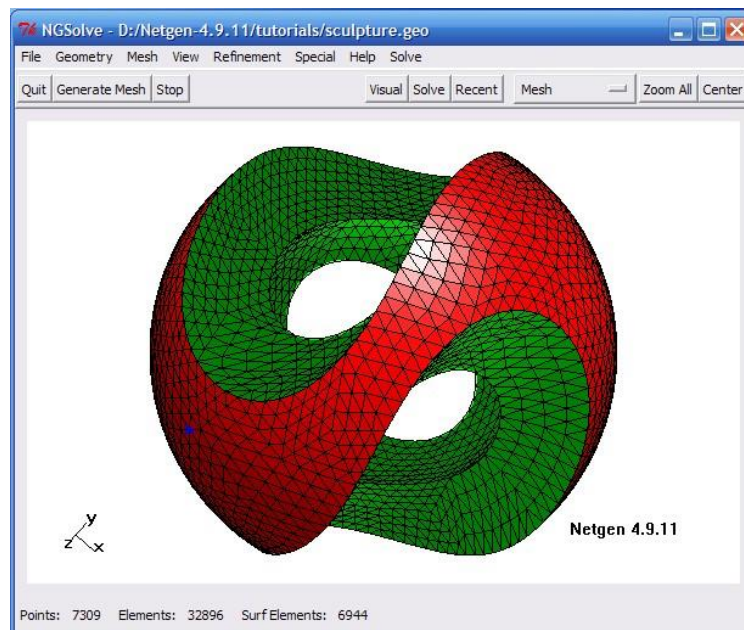


Рис. 1.3. Вікно програми NetGen

У пакеті використовується алгоритм на основі комбінованого методу вичерпання і методу на основі критерію Делоне. Імпорт та експорт даних може бути здійснений через найрізноманітніші формати, в тому числі і через стандартні. Зокрема, NetGen "знає" багато форматів CAD -систем.

Ймовірно, єдиний недолік пакета - неможливість прямого контролю над розмірами елементів.

Автор програми - Joachim Schöberl. Сторінка в Інтернеті: <http://www.hpfem.jku.at/netgen/>.

T3D - програма для побудови якісних сіток методом вичерпування. Складні області дискретизуються методом узгодження по межі. Сама область задається як сукупність поверхонь межі і обмежень. Дані імпортуються та експортуються через текстові файли спеціального формату. Крім того, пакет вміє працювати безпосередньо з системами CAD.

Існують версії пакета для ОС Windows і Linux. Програма володіє консольним інтерфейсом. Для ОС сімейства Unix передбачена можливість візуалізації сіток за допомогою сторонньої програми (Elixir). Також можливий запис сіток в графічних форматах VRML 2.0 і VTK.

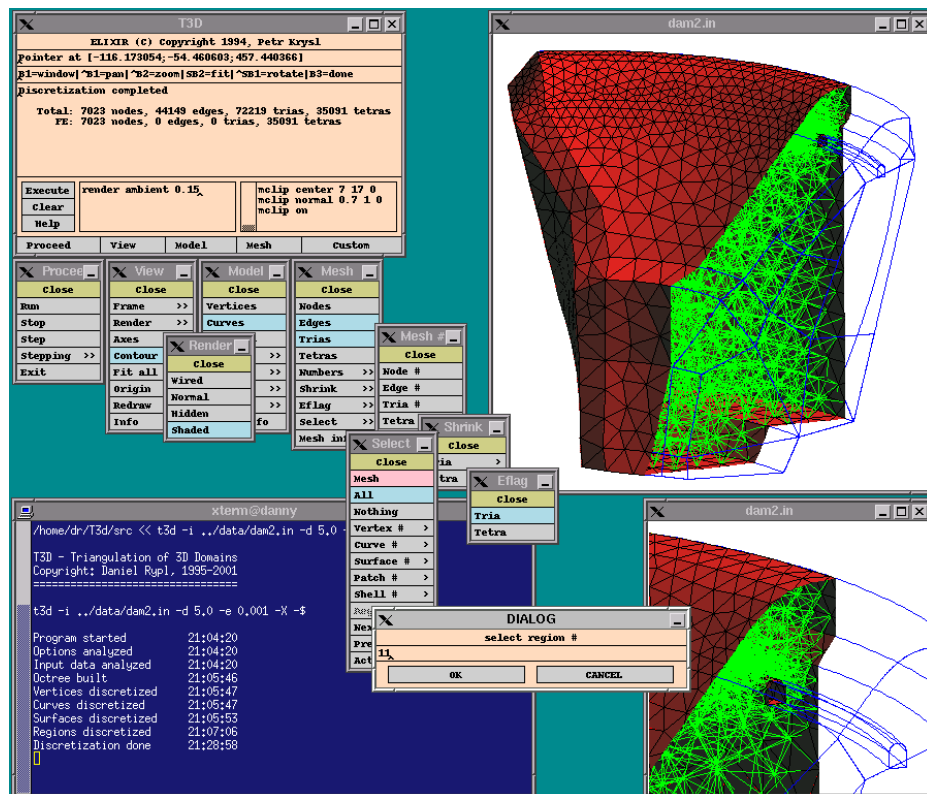


Рис. 1.4. Вікно програми T3D

Пакет безкоштовний для академічного використання, але автором підтримується дуже обмежено. Автор пакета - Daniel Ryp1. Домашня сторінка проекту: <http://ksm.fsv.cvut.cz/Mlr/t3d.h1ml>

1.5. Висновки до розділу

В результаті огляду літературних джерел проаналізовано способи подання геометричних тіл та методи побудови сіток. Розглянуто та проаналізовано критерії якості побудованої сітки. Зроблено огляд сучасних програм для дискретизації просторових конструкцій.

РОЗДІЛ 2

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ТРИВИМІРНОЇ ТРІАНГУЛЯЦІЇ ПРОСТОРОВИХ КОНСТРУКЦІЙ

У цьому розділі буде розглянуто розширення алгоритму рухомого фронту для тривимірного простору. Будуть розглянуті питання його надійності та скінченності алгоритму і складність роботи алгоритму.

Алгоритм рухомого фронту не гарантує побудову тетраедральної сітки для всієї області, тому буде запропонований другий, стійкий алгоритм, заснований на ідеї П.Л. Джорджа [15].

Введемо поняття і позначення. Для чотирьох точок A, B, C, D будемо використовувати наступний скорочений запис:

$$[ABCD] = \det \begin{vmatrix} x_A - x_D & y_A - y_D & z_A - z_D \\ x_B - x_D & y_B - y_D & z_B - z_D \\ x_C - x_D & y_C - y_D & z_C - z_D \end{vmatrix}$$

Орієнтованим трикутником в просторі назвемо трикутник з деяким зафіксованим з точністю до парної перестановки порядком обходу його вершин.

Позитивним півпростором щодо орієнтованого трикутника ABC будемо називати геометричне місце точок X , для яких $[ABCX] > 0$. Аналогічно, негативним півпростором будемо називати геометричне місце точок, для яких $[ABCX] < 0$. Тріангуляцією в просторі назвемо кінцевий набір трикутників, конформно зв'язний через загальні ребра. При цьому два сусідніх трикутника можуть лежати в одній площині. Якщо кожне ребро тріангуляції належить рівно двом трикутникам, то таку тріангуляцію будемо називати замкнутою. Тріангуляцію, трикутники якої не перетинаються один з одним, будемо називати тріангуляцією без само перетинів.

Багатогранником в просторі назвемо обмежену відкриту частину простору, межа якої складається з однієї або декількох непересічних тріангуляцій без само перетинів. Простим багатогранником назвемо багатогранник, обмежений однією замкнутою тріангуляцією. Вершинами

багатогранника будемо називати вершини його тріангуляції, а гранями багатогранника - трикутники тріангуляції.

Грані багатогранника діляться на зовнішні і внутрішні. Причому на зовнішніх гранях можна ввести орієнтацію так, що багатогранник буде лежати в позитивному півпросторі щодо орієнтованого трикутника. Цю орієнтацію будемо називати позитивною.

Фронтом в просторі називатимемо набір орієнтованих трикутників без само перетинів. Кожному багатограннику P можна поставити у відповідність фронт $F(P)$. Для цього введемо на зовнішніх гранях P позитивну орієнтацію, а для кожної внутрішньої грані ABC поставимо у відповідність пару протилежно орієнтованих трикутників ABC і CBA . Сукупність усіх цих орієнтованих трикутників утворює фронт $F(P)$. Будемо називати фронт F замкнутим, якщо існує такий багатогранник P , що $F = F(P)$.

Тетраедральною сіткою T для багатогранної області P назвемо розбиття цієї області на непересічні тетраедри. Будемо називати сітку конформною, якщо будь-які її два елементи або не мають спільних точок, або мають одну спільну вершину, або мають одне загальне ціле ребро, або одну загальну цілу грань. Будемо говорити, що конформна сітка погоджена з межею P , якщо кожна грань P є гранню якогось тетраедра в сітці. Зокрема, у внутрішніх гранях багатогранника буде рівно два сусідніх тетраедра, а у зовнішніх - рівно один.

2.1. Алгоритм рухомого фронту

Відмінною особливістю тривимірного простору є існування таких фронтів, для яких не існує підходящих тетраедрів. З цієї причини в алгоритм додається можливість позначати межі, для яких не знайшлося підходящих тетраедрів, і пропускати їх при подальшому переборі.

В кінці роботи ми отримаємо сітку T для багатогранної під області $P_{\text{mesh}} \subseteq P$ і фронт для нерозбитій багатогранної під області $P_{\text{out}} = P \setminus P_{\text{mesh}}$.

Загальна послідовність дій представлена в Алгоритмі 1.1. Повторимо основні дії, які виконуються при просуванні фронту.

З фронту вибирається непомічена грань ABC з найменшою площею, і будується вершина D у відповідності з обраним параметром s . При реалізації алгоритму в програмному додатку був використаний наступний евристичний метод вибору s і положення вершини D . Нехай a, b, c - довжини сторін трикутника ABC . Спочатку будується центр мас M трикутника ABC , потім від M по нормалі до ABC на відстані $\sqrt{\frac{2}{3}}s$ відкладається вершина D . Параметр s вибирається на підставі функції бажаного розміру елементів, наданої користувачем, або обчислюється автоматично за формулою $s = \gamma \sqrt{\frac{a^2+b^2+c^2}{3}}$, де $\gamma \geq 1$ - деякий параметр, що відповідає за швидкість автоматичного розгублення сітки.

Алгоритм 2.1 Алгоритм рухомого фронту в двовимірному випадку

- 1: Визначимо $T^0 = \emptyset, F^0 = F(P)$.
- 2: для $k = 0, 1, \dots$ початок циклу
- 3: Обрати непомічений $\triangle ABC \in F_k$ з мінімальною площиною.
- 4: Якщо непомічених трикутників не залишилось, то перейти до 22.
- 5: Визначити бажану довжину сторони елемента s .
- 6: Побудувати вершину D з урахуванням s и $[ABCD] > 0$.
- 7: Обрати деякі $R_0 > s, h > 0$ и $r > 0$.
- 8: для $R \in \{R_0, \infty\}$: початок циклу
- 9: Побудувати локальний фронт $\mathcal{F}_R^k: \mathcal{F}_R^k \supset \mathcal{F}^k \cap S_R(D)$
- 10: Визначити множину кандидатів $\Sigma_R^k = \{P_i^k\} \cap S_R(D)$
- 11: Якщо, $\{P_i^k\} \cap S_r(D) = \emptyset$ и $\mathcal{F}^k \cap S_h(D) = \emptyset$, то додати D до Σ_R^k .
- 12: для усіх $X \in \Sigma_R^k$: початок циклу
- 13: Якщо, $ABCX$ не перетинає \mathcal{F}_R^k , тоді
- 14: Додати тетраедр $T_k = ABCX, \mathcal{T}^{k+1} = \mathcal{T}^k \cup \{T_k\}$.
- 15: Оновити фронт: $\mathcal{P}^{k+1} = \mathcal{P}^k \setminus T_k, \mathcal{F}^{k+1} = \mathcal{F}(\mathcal{P}^{k+1})$.
- 16: Перейти до 21.
- 17: кінець якщо
- 18: кінець циклу
- 19: кінець циклу
- 20: Помітити $\triangle ABC$, і перейти до 3
- 21: кінець циклу
- 22: Визначимо $T = T_k, P_{out} = P_k, F_{out} = F_k$.

Оцінимо довжину ребер AD, BD та CD . При $\gamma \geq 1$ висота тетраедра

$s_0 \geq \frac{2a^2+2b^2+2c^2}{9}$. Розглянемо ребро AD: $|AD|^2=|AM|^2+|MD|^2$. Довжина AM виражається з формули для медіани трикутника: $|AM| = \frac{2}{3} \sqrt{\frac{2b^2+2c^2-a^2}{4}}$

$|AM|^2 = \frac{2b^2+2c^2-a^2}{9}$. Таким чином, отримуємо, що

$$|AD|^2 \geq \frac{2b^2 + 2c^2 - a^2}{9} + \frac{2a^2 + 2b^2 + 2c^2}{9} = \frac{a^2 + 4b^2 + 4c^2}{9}$$

Якщо сторони трикутника ABC були менше d, то й |AD| буде не менше d. Аналогічні оцінки вірні для BD і CD. Після вибору положення вершини - кандидата D ми розглядаємо локальну околицю, і для кожного тетраедра - кандидата перевіряємо його перетин з локальним фронтом. Якщо всі кандидати виявилися невідповідними, то проводиться повний перебір всього фронту. Якщо і в цьому випадку відповідного тетраедра знайдено не було, то грань ABC позначається як вже розглянута, і перебір починається з початку для наступної грані.

На відміну від Алгоритму 1.1 ділянка Алгоритму 2.1 з номерами рядків 3-20 може виконатися кілька разів для побудови одного тетраедра.

При кожному виконанні цієї ділянки відповідна грань або видаляється з фронту, або позначається. Наприкінці роботи алгоритму помічені грані і видалені з фронту грані будуть гранями кінцевої сітки T. Тому кількість виконань цієї ділянки алгоритму обмежено зверху не кількістю тетраедрів в кінцевій сітці, а кількістю граней.

Більш детально розглянемо інші відмінності тривимірного алгоритму рухомого фронту від двовимірного аналога.

Алгоритм 2.2 Перевірка перетину трикутника з відрізком в тривимірному просторі

- 1: Знайти загальні вершини у трикутника ABC і відрізка PQ.
- 2: **якщо** загальних вершин немає, **тоді**
- 3: Визначити положення вершин P і Q відносно площини ABC.
- 4: **якщо** P і Q лежать в одному півпросторі, **тоді**
- 5: Перетину немає.
- 6: **інакше якщо** P і Q лежать у різних Напівпростір, **тоді**

- 7: Обчислити $k_A=d(PQBC)$, $k_B=d(PQCA)$, $k_C = d(PQCB)$.
- 8: Якщо серед k_A , k_B , $k_C \in \{1, -1\}$, то перетину немає.
- 9: **інакше якщо** P і Q лежать у площині ABC , **тоді**
- 10: Перевірити, що одна з прямих PQ , AB , BC або CA розділяє трикутник і відрізок в різні півплощини .
- 11: Якщо така пряма знайшлася , то перетину немає.
- 12: **кінець якщо**
- 13: **кінець якщо**
- 14: якщо одна спільна вершина , **тоді**
- 15: Якщо друга вершина відрізка не лежить в площині трикутника, то перетину немає.
- 16: Інакше перевірити що одна з прямих AB , BC , CA розділяє трикутник і відрізок в різні півплощини.
- 17: Якщо така пряма знайшлася, то перетину немає.
- 18: **кінець якщо**
- 19: якщо дві загальних вершини, **тоді**
- 20: Перетину немає.
- 21: **кінець якщо**
- 22: В інших випадках вважаємо, що трикутник і відрізок перетинаються.

Алгоритм 2.3 Перевірка перетину тетраедра з трикутником

- 1: Для кожної грані тетраедра $ABCD$ і ребра трикутника PQR перевірити перетин.
- 2: Для трикутника PQR і кожного ребра тетраедра $ABCD$ перевірити перетин.
- 3: Знайти загальні вершини у тетраедра $ABCD$ і трикутника PQR .
- 4: **якщо** загальних вершин немає, **тоді**
- 5: Якщо три точки P , Q , R лежать всередині $ABCD$, тобто перетин.
- 6: **кінець якщо**
- 7: **якщо** одна спільна вершина, **тоді**
- 8: Якщо дві решта точки трикутника PQR лежать всередині $ABCD$,

тобто перетин.

9: кінець якщо

10: якщо дві загальних вершини, **тоді**

11: Якщо залишилася точка трикутника PQR лежить всередині ABCD,
тобто перетин.

12: кінець якщо

13: В інших випадках вважаємо, що перетину немає.

2.1.1. Перевірка перетину тетраедра з трикутником

Побудуємо алгоритм, аналогічний алгоритму рухомого фронту в двовірному просторі, для алгоритму рухомого фронту в тривірному просторі. Будемо робити перевірки на основі знака визначника матриці 3×3 .

У загальному випадку два опуклих об'єкта не перетинаються тоді, і тільки тоді, коли існує площина, яка розділяє їх. В умовах Алгоритму 2.1 тетраедр і трикутник можуть також мати одну спільну вершину, або одне загальне ребро, або трикутник може бути однією з граней тетраедра. Виділимо з завдання перевірки перетину окрему під задачу перевірки перетину трикутника з відрізком.

Розглянемо загальний випадок.

Будемо шукати таку площину, яка розділяє трикутник і відрізок з урахуванням попереднього зауваження про загальні вершини. Перебір всіх можливих таких площин представлений в Алгоритмі 2.2. В алгоритмі використовується функція $d(ABCD)$, що обчислює знак виразу $[ABCD]$.

Перевірка перетину трикутника з відрізком використовується в Алгоритмі 2.3 перевірки перетину тетраедра і трикутника. Спочатку ми перевіряємо, чи є перетин біля границі тетраедра з границею трикутника, а потім перевіряємо особливі випадки, такі як трикутник, цілком лежить всередині тетраедра.

Запропонована комбінація Алгоритмів 2.2 і 2.3 для перевірки перетину використовує функцію $d(ABCD)$, яка обчислює знак виразу $[ABCD]$.

Нехай про функцію $d(ABCD)$ відомо, що при $d(ABCD) \neq 0$ виконано $d(ABCD) = \text{sign}([ABCD])$. Тоді запропоновані алгоритми виключають виникнення помилок другого типу. Тобто, пересічні насправді тетраедр і трикутник не будуть помилково сприйняті як непересічні. Це гарантує замкнутість і відсутність самоперетинів при просуванні фронту.

2.1.2. Скінченність роботи алгоритму рухомого фронту

Проведемо аналіз Алгоритму 2.1, і покажемо кінцівку числа операцій. У запропонованому алгоритмі три явних вкладених цикли і один неявний за рахунок перезапуску перебору в рядку 20. Цикл перебору в рядку 12 завжди кінцевий чинності кінцівки безлічі Σ_R^k . Цикл в рядку 8 робить не більш двох ітерацій.

У тривимірному випадку існують такі конфігурації фронту, для яких не існує відповідного тетраедра з четвертої вершиною з безлічі вершин - кандидатів фронту. Приклад такої конфігурації фронту, для якої подальше просування неможливо, представлений на рис. 2.1.

Саме з цієї причини в Алгоритмі 2.1, у рядку 20, грані, що не мають відповідних тетраедрів, позначаються і пропускаються при подальшій побудові сітки. Кожна грань позначається не більше одного разу, і кількість граней в F_k скінчено, тому неявний цикл в рядку 20 кінцевий.

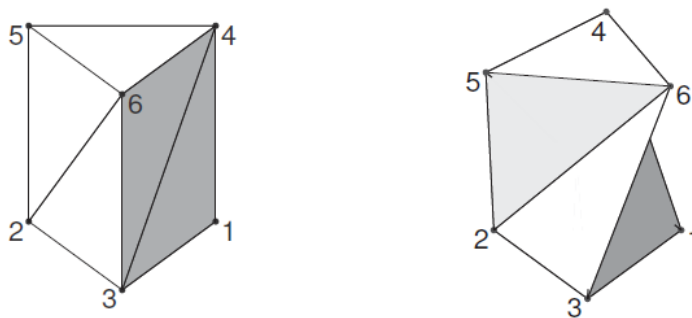


Рис. 2.1. Приклади конфігурації фронту, для яких подальше просування неможливо: призми Шенхардта. На рисунку ребро 1-5 приховано.

Покажемо, що при певному виборі параметрів в Алгоритмі 2.1 можна обмежити максимально можливу кількість побудованих тетраедрів. Ми можемо обмежити знизу відстань між вершинами сітками: $\rho_k > \delta > 0$ при будь-

якому k . Вірна оцінка на максимальну кількість вершин, яку можна вмістити в області з обсягом V , площею поверхні S , і складається з K компонент зв'язності:

$$v_k \leq \frac{6V}{\pi \delta^3} + \frac{3S}{\pi \delta^2} + K.$$

Твердження 2.1.1. Нехай у тривимірному просторі задана довільна сітка з v вершинами, e ребрами, f гранями і n осередками. Тоді вірні такі оцінки:

$$n \leq \max(0, \frac{v(v-3)}{2} - 1), \quad f \leq \max(0, v(v-3)), \quad e \leq \frac{v(v-1)}{2}.$$

Доказ. Використовуємо наступне розширення формули Ейлера:

$$v - e + f - r = k - 1,$$

де v - кількість вершин, e - кількість ребер, f - кількість граней, r - кількість областей, включаючи зовнішню, а k - кількість компонент зв'язності.

Тоді кількість осередків у сітці $n = r - 1$. Нехай $f \geq 4$, тоді у зовнішньої області не менше чотирьох граней. Решта областей є осередками сітки, тому у них теж не менше чотирьох граней у кожній.

Кожна грань належить рівно двом областям, тому $2f \geq 4r$, тобто $r \leq 1/2f$. Складаючи це нерівність з формулою Ейлера, отримуємо $1/2f \leq e - v + k - 1$.

Оцінимо кількість ребер в сітці. У кожній компоненті зв'язності кількість ребер не перевищує кількості різних пар вершин. Нехай у компоненті зв'язності з номером i міститься $v_i \geq 1$ вершин. Тоді

$$e \leq \sum_{i=1}^k \frac{v_i(v_i - 1)}{2}$$

Максимальна кількість ребер при фіксованій кількості вершин і компонент зв'язності досягається у випадку однієї великої компоненти з $v - k + 1$ вершин і $k - 1$ компонент по одній вершині, тобто $e \leq \frac{(v-k+1)(v-k)}{2} \leq \frac{v(v-1)}{2}$.

Позначимо для зручності $d = v - k + 1$, тоді $\frac{1}{2}f \leq e - d$ і $e \leq \frac{d(d-1)}{2}$. Тобто

$$f \leq 2e - 2d \leq d(d-3) \leq v(v-3), \quad \text{і} \quad n = r - 1 \leq \frac{1}{2}f - 1 \leq \frac{v(v-3)}{2} - 1,$$

твердження доведено.

Відзначимо, що ці оцінки дуже "грубі", і на практиці в неструктурованій тетраедральній сітці кількість граней наближено дорівнює $11v$, а кількість тетраедрів наближено дорівнює $5.5v$. З цього твердження

впливає, що кількість тетраедрів, і, отже, число ітерацій зовнішнього циклу в Алгоритмі 2.1 обмежена.

Аналіз складності алгоритму рухомого фронту застосуємо в тривимірному випадку. Оцінка зверху на кількість операцій в гіршому випадку складе величину пропорційну N_F^3 , де N_F - кількість граней в кінцевій сітці, яке обмежене згідно з оцінкою на кількість вершин (2.1) і твердженню 2.1.1. Оцінка в середньому на кількість операцій становить величину, пропорційну $N_F \log(N_F)$. Оцінка середньої швидкості роботи буде експериментально підтверджена в розділі 2.4.1.

Ми показали, що Алгоритм 2.1 зробить кінцеве число операцій. Однак він не гарантує побудову конформної тетраедральної сітки для всієї області P . На практиці алгоритм рухомого фронту розбиває більше 90 % обсягу області, а нерозбита частина області, P_{out} , складається з деякої кількості окремих ланок - багатогранників з малою кількістю граней. Для області P_{out} застосовується другий метод, мова про який піде в наступному розділі.

2.2. Стійкий метод на основі тетраедризації Делоне

У цьому розділі ми запропонуємо метод побудови тетраедральної сітки, узгодженої з заданим фронтом на основі тетраедризації Делоне. Загальна ідея цього методу була запропонована П.Л.Джорджем в [15]. Ми розглянемо спрощену версію цього методу, яку можна застосувати для розбиття ланок, залишених алгоритмом рухомого фронту [18]. Цей метод можна застосовувати для довільних замкнутих фронтів, проте не дозволяє контролювати ні розмір, ні якість побудованих тетраедрів.

Побудова сітки ділиться на три етапи. На першому етапі будується сітка для опуклої оболонки множини точок заданого фронту. На другому етапі сітка подрібнюється і відновлюється геометрія області. На третьому етапі відновлюється узгодженість сітки із заданими фронтом.

2.2.1. Тетраедризація Делоне

Тетраедризацією Делоне для кінцевої безлічі точок $V=\{V_1, \dots, V_n\}$ називається таке конформне розбиття опуклої оболонки множини точок на

тетраедри $T = \{T_1, \dots, T_m\}$, що для будь-якого тетраедра T_i строго всередині описаної навколо нього сфери немає точок з V .

В силу Твердження 2.1.1 кількість тетраедрів m обмежено квадратичною залежністю від числа вершин n .

Існує безліч способів побудови тетраедризації Делоне, ми не будемо зупинятися на деталях, детальніше ознайомитися можна в [19]. Відзначимо лише, що найпростіший ітеративний спосіб побудови по складності в гіршому випадку пропорційний m^2 , і в середньому пропорційний $m^{3/2}$.

2.2.2. Відновлення геометрії області

На цьому етапі у нас є тетраедральна сітка T і заданий фронт F на одному наборі вершині V . Приклад сітки і фронту показаний на Рис.2.2.

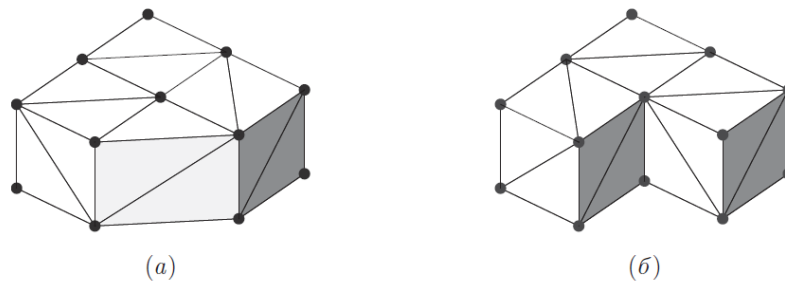


Рис. 2.2. Дві сітки на одному наборі вершин: (а) тетраедральна сітка для опуклої оболонки, (б) фронт F .

Будемо подрібнювати сітку T , додаючи нові вершини на ребрах і гранях, так, щоб ребра і грані фронту F або цілком, або у вигляді своїх розбиттів були представлені в новій сітці. Деталі запропонованого алгоритму представлені в Алгоритмі 2.4.

Алгоритм 2.4 Відновлення геометрії області

- 1: процедура Обробка ребра (V_1, V_2)
- 2: Знайти тетраедр з вершиною V_1 , що перетинає $V_1 V_2$.
- 3: якщо тетраедр знайдений, тоді
- 4: Побудувати точку перетину V на межі тетраедра.
- 5: Розбити відповідні тетраедри в T .
- 6: Обробка ребра (V, V_2) .
- 7: кінець якщо
- 8: кінець процедури

- 9: процедура Обробка граней (V_1, V_2)
- 10: для всіх граней $F \in F$: початок циклу
- 11: якщо грань F перетинає ребро $V_1 V_2$, тоді
- 12: Побудувати точку перетину V .
- 13: Розбити відповідні тетраедри в T .
- 14: Обробка граней (V_1, V).
- 15: Обробка граней (V, V_2).
- 16: Вийти з процедури.
- 17: кінець якщо
- 18: кінець циклу
- 19: кінець процедури
- 20: для всіх ребер $V_1 V_2$ з F : початок циклу
- 21: Обробка ребра (V_1, V_2).
- 22: кінець циклу
- 23: для всіх ребер $V_1 V_2$ з T : початок циклу
- 24: Обробка граней (V_1, V_2).
- 25: кінець циклу
- 26: Видалити тетраедри, що лежать за межами F .

Першим проходом перевіряються перетини ребер фронту F з гранями T . Для кожного ребра з F що перетинають його тетраедри з T розбиваються таким чином, щоб ребро або цілком, або у вигляді розбиття було представлено в сітці. Другий прохід аналогічний, перевіряється перетин ребер сітки T з гранями F . Для кожної грані з F що перетинають її тетраедри з T розбиваються таким чином, щоб грань або цілком, або у вигляді розбиття була представлена в сітці.

Кожен раз при розбитті тетраедрів в T ми створюємо точку перетину, що лежить на F . При цьому розбиваються тільки елементи з T . Наприкінці F залишиться без змін, а нова сітка T своїми гранями буде повністю покривати грані F .

Видаляючи тетраедри з T , що лежать за межами багатогранника P , ми відновимо геометрію області. На рис. 2.3 показана подрібнена сітка T до і після видалення тетраедрів.

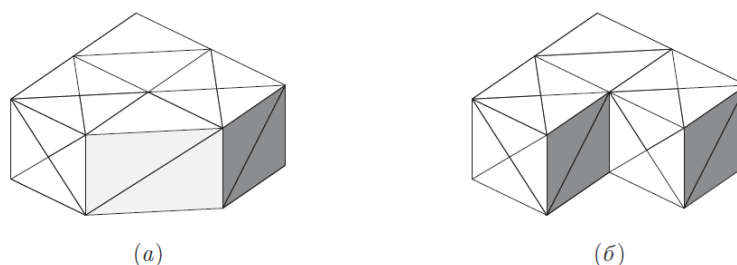


Рис. 2.3. Відновлення геометрії області: (а) подрібнена сітка, (б) сітка з правильною геометрією області.

Отримана сітка T буде конформною сіткою для багатогранної області P , однак вона не буде узгоджена з його межею F . На межі T лежатимуть нові точки перетину, які ми тільки що побудували (порівняйте Рис. 2.2, б і Рис. 2.3, б), алгоритм видалення цих точок з межі буде представлений в наступному розділі.

2.2.3. Відновлення сліду сітки на межі

На завершальному етапі з межі сітки видаляються зайві вершини. Це досягається за рахунок зсуву їх всередину області, та заповнення утворених "вм'ятин" конформними тетраедральними сітками.

Всі зайві точки діляться на точки, що лежать на ребрах F , і на точки, що лежать на гранях F . Позбавлення від точок в обох випадках відбувається одним і тим же способом. Розглянемо для визначеності перший випадок.

Нехай P - точка на граничному ребрі, яку треба прибрати з поверхні. Припустимо для простоти викладу, що фронт F є простим.

З незначними доповненнями запропонований алгоритм застосуємо і в загальному випадку.

Розглянемо безліч Σ_P тетраедрів з T з вершиною P . Межа цієї множини, $\Omega_P = \partial \Sigma_P$, складається з трикутників двох типів: що лежать на межі P , і лежать всередині P . При зсуві вершини P всередину P на межі області утворюються "вм'ятини". Ці вм'ятини насправді є об'єднанням двох конусів з вершиною P і багатокутними підставами.

Кожен конус може бути конформно розбитий на тетраедри, для цього багатокутні підстави розбиваються діагоналями на трикутники.

Можна показати, що всередині Σ_P завжди є непорожнє відкрита безліч точок, куди може бути зрушена вершина P при збереженні невироджені тетраедрів сітки T . На рис. 2.4 представлена сітка T з вм'ятинами і з закладеними вм'ятинами.

Другий випадок, коли вершина лежить всередині грані F розглядається аналогічно, єдина відмінність - замість двох конусів вм'ятина буде одним конусом з багатокутною межею.

Повний алгоритм дій на цьому етапі представлений в Алгоритмі 2.5.

Спочатку розглядаються вершини на ребрах F , а потім на гранях F .

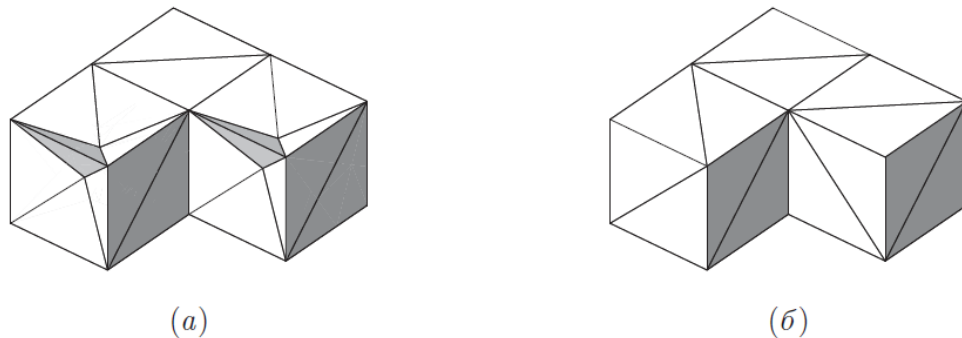


Рис. 2.4. Відновлення сліду сітки на межі: (а) сітка з вм'ятинами, (б) сітка з правильним слідом на межі.

Алгоритм 2.5 Відновлення сліду сітки на межі

- 1: для всіх вершин P на ребрах F і гранях F : початок циклу
- 2: Побудувати безліч сусідніх тетраедрів Σ_P .
- 3: з $\Omega_P = \partial \Sigma_P$ виділити грані, що лежать на F : $\Omega_P = \Omega_P \cap F$.
- 4: перерозбити Ω_P на трикутники, виключивши вершину P .
- 5: Додати до T тетраедри з тріангуляції Ω_P і вершини P .
- 6: Зрушити P всередину Σ_P , відновивши невироджені тетраедри.
- 7: кінець циклу

Запропонований метод гарантує побудову сітки, узгодженої з заданим замкнутим фронтом, за умови, що всі обчислення проводяться точно. Якість побудованих тетраедрів ніяк не обмежується знизу. На практиці через

накопичення обчислювальних похибок метод може побудувати вироджені або вивернуті тетраедри. З цієї причини в якості третього етапу необхідне застосування алгоритмів, що дозволяють значно поліпшити якість сітки і позбутися від вироджених елементів.

2.2.4. Скінченність роботи алгоритму

Проведемо короткий аналіз кінцівки числа операцій в запропонованому методі, не вдаючись у оцінку обчислювальної складності.

На першому етапі ми будуємо тетраедризацію Делоне Т. Кількість вершин в ній збігається з кількістю вершин в F. При наявності оцінки на кількість вершин у сітці Т за допомогою Твердження 2.1.1 ми отримуємо оцінки на кількість ребер, граней і тетраедрів. На другому етапі при відновленні геометрії в Алгоритмі 2.4 використовуються три основних операції. У рядку 21 для кожного ребра з F нових вершин в Т може бути додано не більше ніж кількість тетраедрів в Т на момент перед виконанням рядка 21. Кількість ребер в F звичайно, і отже, обмежено кількість нових вершин. У рядку 24 для кожного ребра з Т нових вершин в Т додається не більше, ніж кількість граней в F. Відзначимо, що нові ребра при додаванні в Т будуть лежати на F, і тим самим самі не будуть породжувати нових вершин.

Решта операції, видалення зовнішніх тетраедрів в рядку 24 Алгоритму 2.4 і зрушення зайвих точок в межі області в Алгоритмі 2.5, не додають нових вершин в Т, і тим самим будуть працювати кінцевий час.

На практиці запропонований алгоритм на основі тетраедризації Делоне використовується тільки для локалізованих ланок з невеликою кількістю вершин і граней. Кількість побудованих з його допомогою тетраедрів невелике, і час роботи робить незначний внесок у загальний час роботи всього ланцюжка побудови тетраедральних сіток.

2.3. Поліпшення якості отриманої сітки

Для побудови тетраедральної сітки використовується комбінація трьох алгоритмів. Перший алгоритм рухомого фронту використовується для побудови більшої частини сітки з можливістю контролювання кроку сітки та

якості тетраедрів. Другий алгоритм на основі тетраедрізації Делона використовується для розбиття частини, що залишилася області. Третій, заключний алгоритм використовується для поліпшення якості сітки.

Відзначимо, що для досягнення хорошої якості сітки з використанням тільки перших двох алгоритмів, як правило, необхідна хороша поверхнева сітка - початковий фронт. Хороший початковий фронт, відповідним чином підібрана функція, що відповідає за бажаний розмір тетраедрів, або правильно підібраний параметр автоматичного розгублення є ключовими факторами для отримання якісної сітки.

У рамках повністю автоматичної і надійної технології побудови неструктурованих тетраедральних сіток необхідно додаткове поліпшення якості сітки за допомогою третього алгоритму.

Найпростіші методи використовує зсув вершин сітки без зміни топологічної структури. Використання локальних модифікацій топології сітки, таких як розбиття ребра новою вершиною, стягнення ребра в одну вершину і др., на додаток до зрушення вершин як правило дозволяє значно поліпшити якість сітки.

2.3.1. Алгоритми оптимізації сіток

Оптимізаційний процес являє собою інтерактивне застосування різних методів, геометричних і топологічних, які виконуються над поверхневою сіткою. Реалізація цих методів на «низькому рівні» зводиться до наступних базових операцій:

- 1) Зміна координат точки.
- 2) Додавання вузла в сітку.
- 3) Видалення вузла з сітки.
- 4) Видалення ребра з сітки.
- 5) Обмін ребер.

2.3.1.1. Алгоритми виконання базових операцій перебудови сіток

Зміна координати точки не впливає на топологію сітки, тому ця операція не впливає на відносини, а впливає тільки на координатні масиви. Відповідно,

ми цією операцією міняємо для i -ї точки значення координатних масивів x_array , y_array і z_array .

Операція додавання вузла в сітку пов'язана з операцією розбиття ребра, тому нам необхідно знати, на якому ребрі буде додано вузол.

Алгоритм додавання вузла виглядає наступним чином:

- Знаходимо індекси точок, які складають вихідне ребро e_1 (p_1, p_2).
- Знаходимо індекси трикутників, суміжних з вихідним ребром (t_1, t_2).
- Знаходимо індекси точок, які належать цим трикутникам, але не лежать на ребрі (p_3, p_4).
- Знаходимо індекси всіх ребер даних трикутників, за винятком вихідного ребра (e_2, e_3, e_4, e_5)
- Заводимо нові індекси в масивах елементів. Додаються одна точка p_5 , два трикутника t_3, t_4 і три ребра e_6, e_7, e_8 .
- Змінюємо всі наявні відносини відповідно з новими елементами.
- Додаємо в координатні масиви значення координати нової точки.

Операція видалення вузла з сітки пов'язана з видаленням ребра. Вона змінює координатні масиви, виробляючи послідовний зсув значень. Відповідно до нових значеннями індексів змінюються значення елементів записів у відносинах.

Видалення ребра з сітки викликається в методі об'єднання трикутників.

Ця операція проводиться за наступним алгоритмом:

- Знаходимо для вихідного ребра e інцидентне йому точки p_1 і p_2 .
- Знаходимо індекси трикутників, суміжних з вихідним ребром.
- Знаходимо індекси точок, які належать цим трикутникам, але не лежать на ребрі (p_3, p_4).
- Знаходимо індекси всіх ребер, інцидентних з вершинами p_1 і p_2 .
- Знаходимо індекси всіх трикутників, інцидентних з вершинами p_1 і p_2 .
- Змінюємо всі наявні відносини відповідно до елементів, які видаляються: одне ребро (e), одна точка (p_2) і два трикутника (t_1, t_2).

- Видаляємо точку з сітки.

Обмін ребер - операція, яка не торкає координатні масиви, але змінює відносини. Реалізована вона наступним чином:

- Вибираємо одне ребро з міняних за вихідне.
- Знаходимо для вихідного ребра е інцидентні йому точки p_1 і p_2 .
- Знаходимо індекси трикутників, суміжних з вихідним ребром.
- Знаходимо індекси точок, які належать цим трикутникам, але не лежать на ребрі (p_3, p_4).
- Знаходимо індекси всіх ребер даних трикутників, за винятком вихідного ребра (e_2, e_3, e_4, e_5).
- Змінюємо всі наявні відносини відповідно з новими індексами.

2.3.1.2. Алгоритми основних операцій оптимізації

Для збереження точності апроксимації поверхневих особливостей ми вводимо так звану функцію гостроти $Sh()$, яка визначається для ребер і вершин сітки і використовується при виконанні всіх процедур локальних перебудов сіток для виявлення і збереження гострих кутів і ребер.

Для сіткових ребер e , функція $Sh(e)$ задається у вигляді:

$$Sh(e) = \begin{cases} 1, & \text{якщо } \frac{(1 + (\bar{n}^T(t_1) * (\bar{n}^T(t_2)))}{2} \leq \varphi \text{ і } \frac{(1 + (\bar{N}^T(t_1) * (\bar{N}^T(t_2)))}{2} \leq \varphi \\ 0, & \text{в іншому випадку} \end{cases} \quad (2.1)$$

Тут $\bar{n}(t_1), \bar{n}(t_2)$ - одиничні нормалі до сусідніх трикутників t_1, t_2 , для яких e - загальне ребро. $\bar{N}(t_1), \bar{N}(t_2)$ - нормалі до вихідної поверхні в центрах мас трикутників t_1, t_2 ; φ - деяке порогове значення. Нормалі до поверхні обчислюються як нормалізоване градієнт функції, що описує вихідну поверхню V_F . Ненульове значення $Sh(e)$ означає, що сіткове ребро e лежить на гострому ребрі поверхні V_F .

На основі базових операцій ми реалізуємо операції оптимізації. Тобто згладжування сітки, переміщення точок у напрямку до вихідної поверхні, обмін ребер, розбиття ребер і об'єднання трикутників.

Згладжування сітки ми проводимо наступним чином:

- Для всіх ребер обчислюємо функцію гостроти.

- Точки, які належать гострим ребрам з $Sh(e)=1$, маркуємо як непереміщуваними.
- Для всіх точок послідовно згідно з формулою обчислюємо λ як $l_{min}/10$, вектор напрямку руху \bar{U}_i .
- Якщо точка переміщується, змінюємо її координати на нові.

Алгоритм переміщення точок у напрямку до вихідної поверхні був реалізований в двох версіях. Перша являє собою тільки ітеративне застосування формули (2.1) послідовно на всі точки.

У другій версії ми ітеративно виконуємо наступний процес:

- Якщо точка непереміщується, то пропускаємо.
- Для точки вважаємо оптимальну довжину переміщення по напрямку до поверхні, варіюючи значення r в (2.1) від 0.01 до 0.1.
- Якщо точка вийшла на поверхню, тобто $\sigma_v < \varepsilon$, то маркуємо точку яка непереміщується.
- Міняємо координати точки на нові.

Обмін ребер сусідніх трикутників для усунення вироджених трикутників проводиться таким чином:

- Проходимо послідовно по всіх трикутниках сітки.
- Якщо в трикутнику всі кути менше встановлених користувачем, то пропускаємо.
- Для великого кута знаходимо індекс ребра, протилежного кута.
- Знаходимо другий трикутник, суміжний з вихідним по цьому ребру.
- Перевіряємо, які кути будуть після операції обміну ребер. Якщо відбувається погіршення - пропускаємо трикутник.
- Обчислюємо нормалі для трикутників до і після операції обміну ребер.
- Якщо відбувається погіршення відхилення нормалей (в рамках задаються користувачем значень), пропускаємо трикутник.

- Викликаємо базову операцію обміну ребер.

Подібним чином проходить операція з обміну ребер з метою поліпшення відхилення нормалі в центроїді трикутника від нормалі до вихідної поверхні:

- Проходимо послідовно по всіх ребрах сітки.
- Для ребра знаходимо два пов'язаних з ним трикутника.
- Якщо ребро входить більш, ніж у два трикутника (випадок «захльосту»), то пропускаємо ребро.
- Обчислюємо відхилення ε_v і ε_n . ε_v вираховується як відхилення точки в середині ребра від поверхні об'єкту. ε_n - відхилення нормалі трикутника від нормалі вихідної поверхні, обчислюється за формулою (2.1).
- Якщо відбувається погіршення ε_v і ε_n - пропускаємо ребро.
- Викликаємо базову операцію обміну ребер.

Розбиття ребер відбувається за наступним алгоритмом:

- Сортуємо вершини залежно від обраного критерію (посилання на формулу).
- Проходимо послідовно по всіх ребрах сітки.
- Послідовно обчислюємо всі кількісні оцінки (посилання на формулу).
- Якщо умови розбиття задовільні, то розбиваємо ребро.
- Отриману точку виводимо на поверхню шляхом виклику для цієї точки операції переміщення точок у напрямку до вихідної поверхні.

Для об'єднання ребер використовується наступна послідовність дій:

- Проходимо послідовно по всіх ребрах сітки.
- Якщо ребро входить більше, ніж у два трикутника (випадок «захльосту»), пропускаємо це ребро.
- Якщо ребро менше норми L_{min} , тоді проводимо об'єднання трикутників по ребру без додаткових перевірок.
- Розраховуємо функцію гостроти для точок, які належать ребру.

- Розраховуємо вагові коефіцієнти для точок в залежності від функції гостроти.
- Залежно від вагових коефіцієнтів робимо висновок - видаляти або не видаляти.
- Якщо видаляти, виробляємо базову операцію об'єднання двох трикутників по ребру.

2.4. Результати експериментів роботи алгоритмів

Для обчислення якості елементів сітки ми будемо використовувати наступну формулу для якості тетраедра ABCD:

$$q_T = 36\sqrt{2} \frac{V}{\sum l_{ij}^3}$$

де $V = \frac{1}{6}[ABCD]$, l_{ij} - довжини ребер тетраедра. Величина q_T для не вироджених тетраедрів лежить в інтервалі $(0,1]$. Для рівностороннього тетраедра $q_T=1$. Для вироджених або вивернутих тетраедрів $q_T \leq 0$.

Алгоритм рухомого фронту завжди будує тетраедри з $q_T > 0$. При обчисленні якості тетраедра, побудованого з алгоритмом на основі тетраедризації Делоне, через неточні обчислення може вийти $q_T \leq 0$.

По відношенню до всієї сітки нас буде в першу чергу цікавити найгірша якість тетраедрів q_{\min} і розподіл якості по всій сітці. Для кращого відображення результатів будемо використовувати логарифмічну шкалу. Експерименти проводилися на різних машинах при різних умовах. Однак тестові запуски в межах одного експерименту проводилися в однакових умовах.

2.4.1. Швидкість роботи алгоритму рухомого фронту

Експериментально виміряємо швидкість роботи алгоритму рухомого фронту. В якості області візьмемо одиничний куб. На поверхні куба за допомогою алгоритму рухомого фронту, який описано в розділі 2.3, будується квазірівномірна сітка з кроком h . Отримана поверхнева тріангуляція використовується як початковий фронт для тривимірного алгоритму рухомого фронту для побудови квазірівномірної сітки з кроком h .

Зменшуючи крок сітки h , будемо стежити за кількістю тетраедрів в сітці, N_T , і часом побудови сітки, t . Результати експериментів представлені в таблиці 2.1.

Таблиця 2.1

**Швидкість роботи алгоритму рухомого фронту
в тривимірному просторі**

h	N_T	t , сек	$N_T/1$ сек	$t/(N_T \log N_T)$, МКС
0.1	3544	0.48	7383	16.6
0.05	27988	4.81	5819	16.8
0.025	204734	43.35	4723	17.3
0.0125	1613980	406.75	3968	17.6

З результатів розрахунків видно, що час роботи пропорційний величині $N_T \log N_T$. У таблицю також включені стовпці зі швидкістю побудови (тетраедри в секунду), і оцінкою коефіцієнта перед $N_T \log N_T$.

2.5. Висновки до розділу

Розроблено, реалізовано і протестовано технологію побудови конформних тетраедральних сіток, узгоджених із заданою дискретною межею, для багатокомпонентних та багатограних областей. Доведено кінцівку числа операцій і коректність роботи всього ланцюжка за умови використання точних обчислень. Доведено коректність роботи алгоритму рухомого фронту. Експериментально підтверджена оцінка складності роботи алгоритму рухомого фронту в середньому.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ ТРИВИМІРНОЇ ТРИАНГУЛЯЦІЇ ПРОСТОРОВИХ КОНСТРУКЦІЙ

3.1. Об'єктно-орієнтована модель розрахункової сітки

Повний проєкт програмної системи являє собою сукупність моделей логічного і фізичного уявлень, які повинні бути узгоджені між собою. У мові UML для статичного представлення моделей систем використовуються діаграми класів. Вони визначають склад класів об'єктів і їх зв'язків.

На діаграмі класів (Рис.3.1.) прямокутники, поділені на три частини – це класи, лінії – зв'язки між ними. Ім'я класу записано в верхній частині прямокутника. В другій і третій частині – відповідно список атрибутів і операції, що мають специфікатори доступу.

Специфікація класів препроцесора.

cPoint- клас вершин елементів в n -мірному просторі, який містить координати точки, її індивідуальний номер в списку вершин усіх точок.

cElement- клас елементів в просторі R_n . Цей клас зберігає в собі список номерів вершин, їх кількість, індивідуальний номер елементу, координати його центру мас, об'єм, номери сусідніх елементів, що мають з ним загальну грань. Для моделювання завдань, що мають об'єкти з різними фізичними властивостями використовується змінна, що містить номер підобласті, якій належить елемент. Крім того клас елементу містить методи для динамічного обчислення граней.

cFacet- клас граней елементу, який містить список номерів вершин, що утворюють грань; їх кількість, площу, нормаль, орієнтовану з елементу хазяїна грані, номер елементу сусіда хазяїна грані, і номер елементу хазяїна грані. Грань обчислюється динамічно у міру використання для економії пам'яті.

cMesh- клас для зберігання і доступу до сітки в тілі програми. Цей клас містить список точок і елементів сітки. Іноді такі класи містять список граней. Проте, як показує практика, зберігання списків граней призводить до значної перевитрати пам'яті. Цей клас містить у тому числі методи для: читання і

запису сіток в різних сіткових форматах, методи для реалізації згущення і розрідження сіток, які використовуються для адаптивних сіткових методів, методи по перебудові сіток.

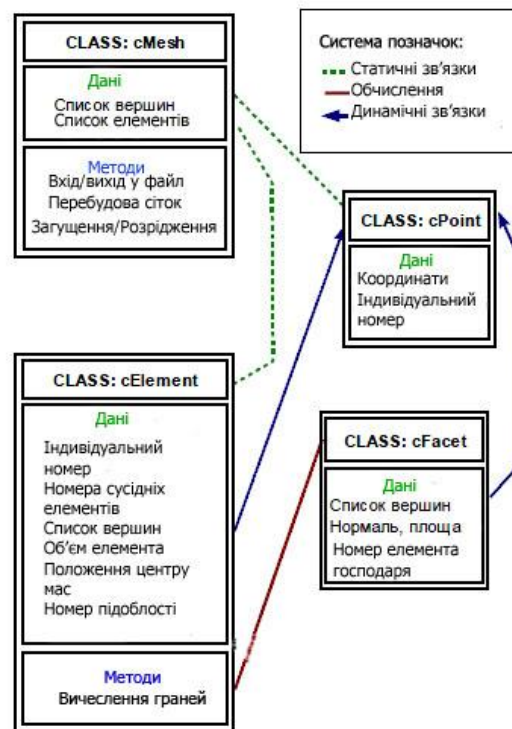


Рис. 3.1. Діаграма класів препроцесора

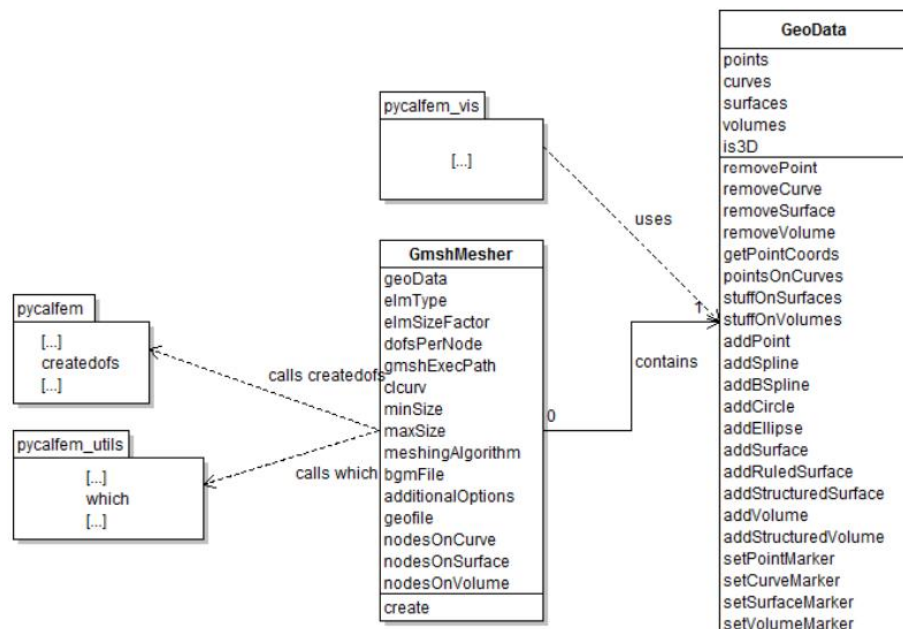


Рис.3.2. Діаграма класів нових класів

Квадрати з вкладками вгорі є модулями. Показано «публічні» методи, атрибути та функції, які викликаються за класами.

Обчислювальна сітка міститься в об'єкті `chi_mesh::MeshContinuum`, який можна отримати за допомогою:

```
auto grid_ptr = cur_handler->GetGrid();
```

Детальніше про структури даних Mesh

chi_mesh::MeshHandler

Меши та операції з сітками обробляються `chi_mesh::MeshHandler`. Обробники сітки завантажуються в глобальну змінну `chi_meshhandler_stack`, а поточний обробник відстежується іншою глобальною змінною, `chi_current_mesh_handler`. Якщо виконується будь-яка операція з сіткою, вона повинна поміщати новостворені об'єкти у стек в обробнику. На рисунку 3.3 показано узагальнену архітектуру.

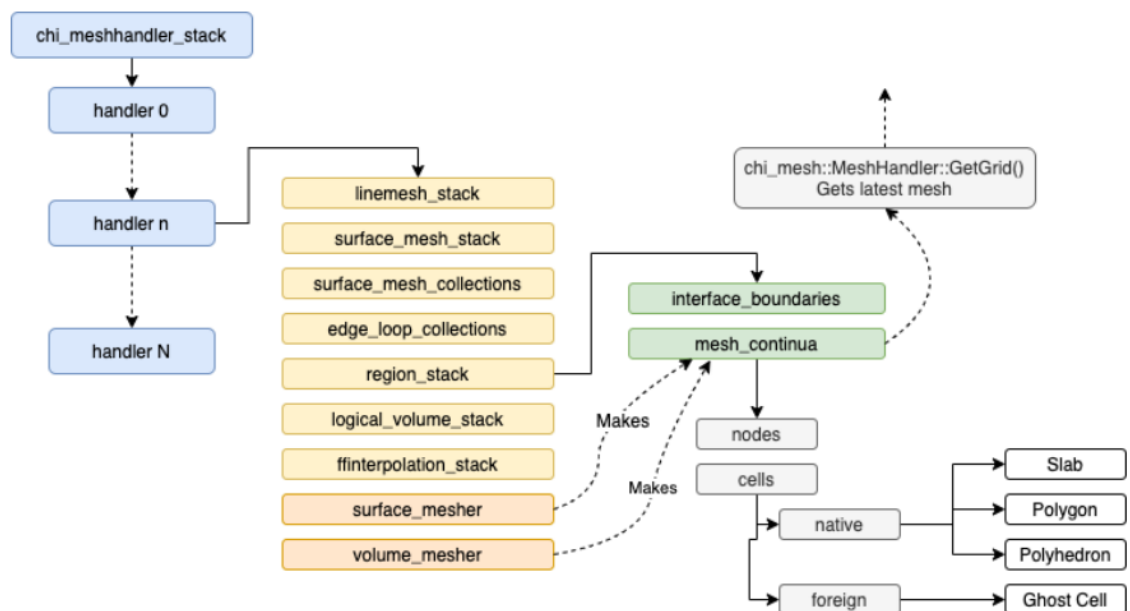


Рис. 3.3. Огляд ієрархії сітки

Кожен обробник сітки обладнано однією сіткою для поверхні та однією сіткою для об'єму, обидві спочатку невизначені. Етап сітки поверхні можна розглядати як етап попередньої обробки.

Види поверхневих сіток:

chi_mesh::SurfaceMesherPredefined. Пройти через препроцесор. Не виконує жодного зчеплення.

chi_mesh::SurfaceMesherDelaunay. Змінює поверхневу сітку за допомогою тріангуляції Делоне.

chi_mesh::SurfaceMesherTriangle. Змінює сітку поверхні за допомогою Triangle 1.6.

Так само існують різні типи об'ємних сіток:

chi_mesh::VolumeMesherLinemesh1D. Перетворює лінійні сітки на сляби.

chi_mesh::VolumeMesherPredefined2D. Перетворює попередньо визначені сітки поверхонь у двовимірні трикутники, чотирикутники або багатокутники.

chi_mesh::VolumeMesherExtruder. Екстрадує попередньо визначені поверхневі сітки в тривимірні трикутні призми, шестигранники або багатогранники.

chi_mesh::VolumeMesherPredefined3D. Перетворює завантажені 3D-сітки на 3D-тетраедри, шестигранники або багатогранники.

chi_mesh::VolumeMesherPredefinedUnpartitioned. Перетворює легку нерозділену сітку на правильну розділену сітку з повними деталями. Цей мешер забезпечує велику гнучкість для читання сіток із зовнішніх джерел.

Поверхневі сітки та об'ємні сітки призначаються обробнику як:

```
cur_handler->surface_mesher = new chi_mesh::SurfaceMesherPredefined;  
cur_handler->volume_mesher = new chi_mesh::VolumeMesherPredefined3D;
```

Щоб виконати поверхневі сітки, просто виконайте:

```
cur_handler->surface_mesher->Execute();  
cur_handler->volume_mesher->Execute();
```

Комірки в Chi-Tech є основними будівельними блоками для наукових обчислень на основі сітки. Деякі типи сіток показані на рисунку 3.4 та визначаються переліком, який вони містять, як визначено **chi_mesh::CellType**. Наразі підтримуються такі типи комірок:

- **chi_mesh::CellType::SLAB**
- **chi_mesh::CellType::TRIANGLE**
- **chi_mesh::CellType::QUADRILATERAL**
- **chi_mesh::CellType::POLYGON**

- `chi_mesh::CellType::TETRAHEDRON`
- `chi_mesh::CellType::HEXAHEDRON`
- `chi_mesh::CellType::POLYHEDRON`

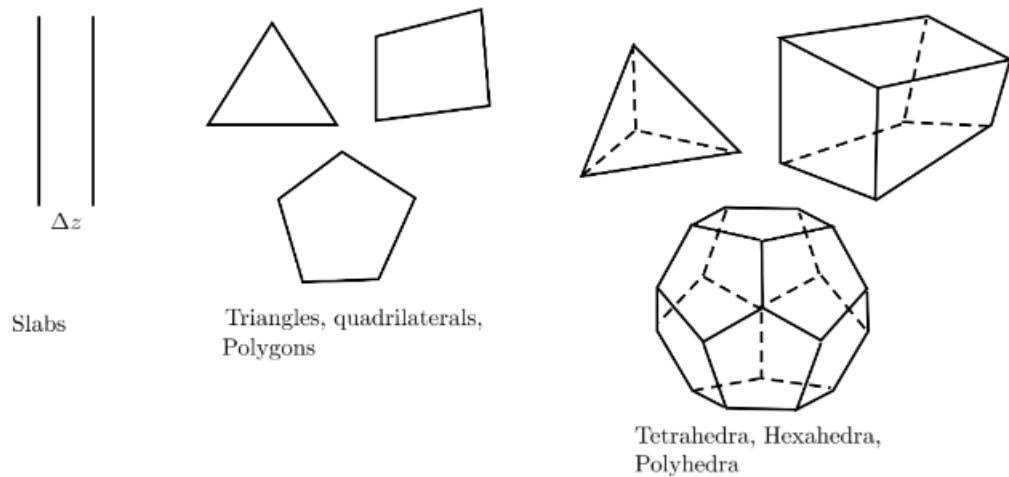


Рис. 3.4. Типи клітин

Клітини живуть у члені `local_cells` сітки, яка має тип об'єкта `chi_mesh::MeshContinuum::LocalCells` під егідою `native_cells` або `external_cells`. Гарантовано, що локальні клітини є повністю визначеними, тоді як нелокальні клітини, швидше за все, будуть клітинами-привидами. Найшвидший спосіб отримати доступ до комірки за допомогою її локального ідентифікатора.

```
auto cell = grid->local_cells[cell_local_id];
```

Об'єкт `local_cells` також сприяє ітератору.

```
for (auto cell = grid->local_cells.begin();
```

```
cell != grid->local_cells.end();
```

```
++cell)
```

```
{ //do stuff }
```

а також ітератор на основі діапазону

```
for (auto& cell : grid->local_cells)
```

```
{ //do stuff }
```

Крім того, більш дорогий спосіб отримати доступ до комірки за допомогою її глобального індексу через елемент клітинки сітки. Цей член є допоміжним об'єктом, який шукатиме в `native_cells` і `external_cells` клітинку з пов'язаним глобальним ідентифікатором

```
auto cell = grid->cells[cell_global_id];
```

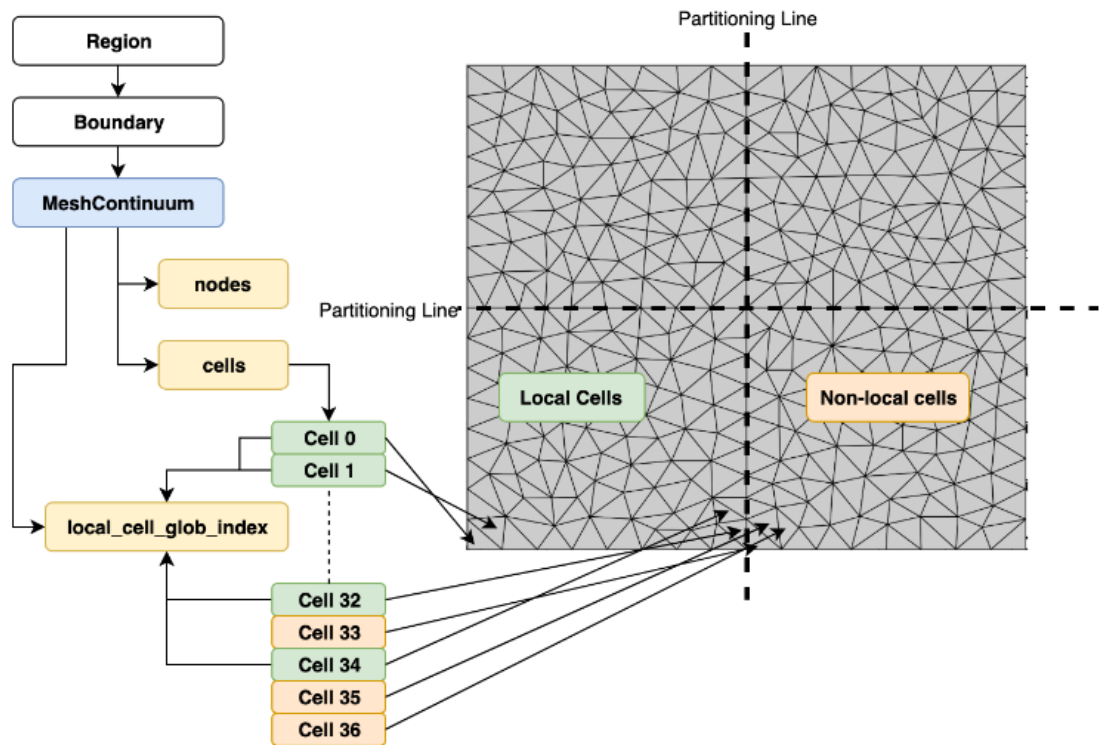


Рис. 3.5. Логіка відображення клітинок

3.2. Обґрунтування вибору середовища розробки програмного додатку

Microsoft Visual C++ (MSVC) — інтегроване середовище розробки додатків на мові C++, що розроблене фірмою Microsoft і поставляється як частина комплексу Microsoft Visual Studio 2022.

Visual C++ 2022 надає потужне і гнучке середовище розробки, що дозволяє створювати додатки для Microsoft Windows і додатки, засновані на Microsoft .NET. Це середовище можна використовувати як інтегроване середовище розробки, так і в якості окремих засобів. Visual C++ складається з наступних компонентів:

- **Компілятор C++:** Visual C++ постачається з компілятором, який перетворює код на мові C++ у машинний код, який може бути виконаний операційною системою.
- **Інтегроване середовище розробки (IDE):** Visual C++ має потужне інтегроване середовище розробки (IDE) під назвою Visual Studio. Це середовище надає розширені можливості для

розробки, налагодження, тестування та керування проектами на мові C++. Воно має інтерфейс користувача з багатьма функціями, такими як редактор коду, відладчик, система контролю версій, інструменти тестування та багато іншого.

- **Бібліотеки Windows API:** Visual C++ надає доступ до багатьох бібліотек Windows API, які дозволяють розробникам взаємодіяти з операційною системою Windows. Ці бібліотеки надають функції для керування вікнами, обробки подій, мережевого взаємодії, роботи з файлами та багато іншого.
- **Бібліотеки стандарту C++:** Visual C++ включає бібліотеки стандарту C++, такі як STL (Standard Template Library), які надають реалізації різних алгоритмів, контейнерів та інших корисних компонентів. Ці бібліотеки полегшують розробку програм на мові C++ і забезпечують переносимість коду між різними платформами.
- **Інструменти для налагодження та профілювання:** Visual C++ надає різноманітні інструменти для налагодження та профілювання програм. Це включає точковий налагоджувач (debugger) для виявлення та виправлення помилок, аналізатор пам'яті для виявлення витоків пам'яті.
- **Пакети розробки (SDK):** Visual C++ має різноманітні пакети розробки (SDK), які дозволяють розробникам створювати програми для конкретних платформ або функціональностей. Наприклад, Windows SDK надає набір інструментів та бібліотек для розробки програм під операційну систему Windows.
- **Інструменти для розробки графічного інтерфейсу:** Visual C++ має набір інструментів для розробки графічного інтерфейсу користувача. Включаючи дизайнер форм, який дозволяє створювати і налаштовувати вікна, кнопки, поля введення та інші елементи інтерфейсу.

Ці компоненти разом створюють потужне середовище розробки для програм на мові C++, дозволяючи розробникам створювати різні типи програм з використанням широкого набору інструментів та бібліотек.

Мова C++, що є найпопулярнішою у світі мовою рівня системи, і Visual C++ разом надають розробникові висококласний засіб світового рівня для побудови програмного забезпечення.

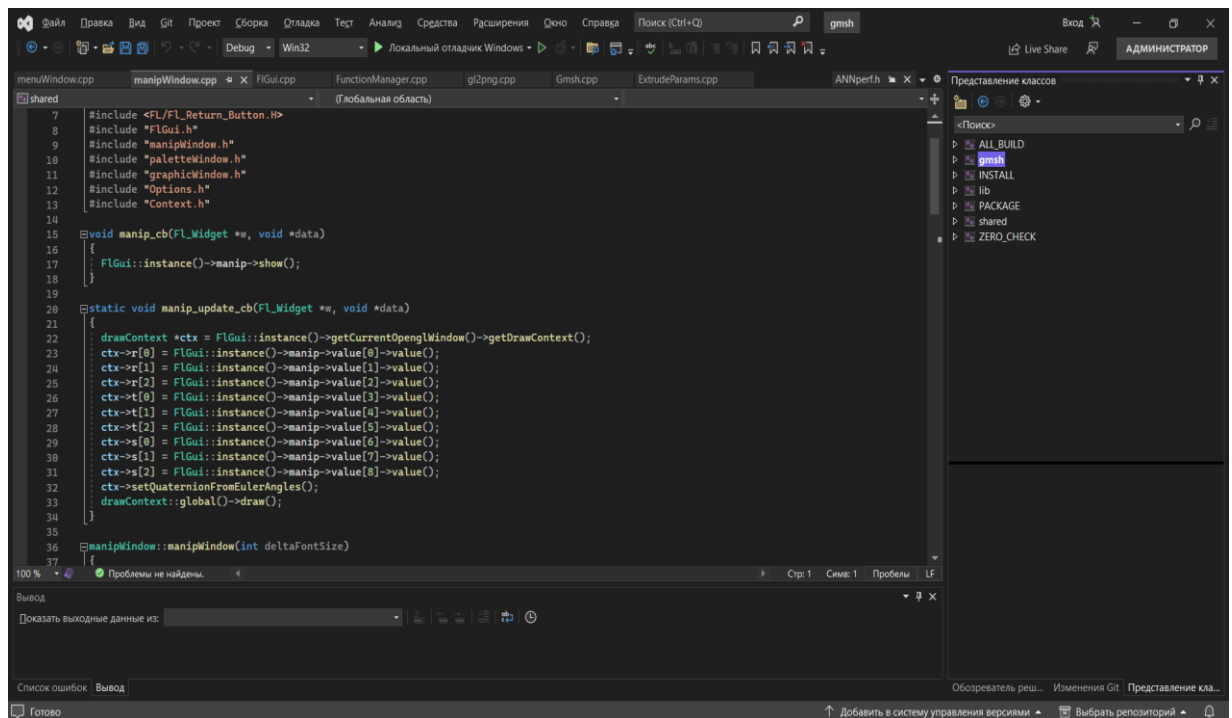


Рис. 3.6. Microsoft Visual Studio 2022

Програма реалізована на мові програмування C++ в середовищі Visual C++ 2022. Вибір середовища програмування обумовлений відносною простотою реалізації графічного інтерфейсу користувача (GraphicUserInterface – GUI).

При створенні засобів відображення і візуалізації використаний графічний інтерфейс OpenGL.

OpenGL має основну мету - відображення статичних та динамічних сцен з двовимірними та тривимірними об'єктами. У цьому контексті, об'єкти можуть бути представлені як набір вершин (для геометричних фігур) або пікселів (для растрових зображень). Починаючи з вихідних даних, OpenGL виконує перетворення, щоб перетворити примітиви та зображення в піксельне представлення. Кожен сформований піксель асоціюється з необхідними

даними для його відображення та подальшої обробки, а результат перетворення розміщується в буфері кадру. Реалізація OpenGL для Windows включає понад 400 функцій, з яких 368 є базовими функціями основної бібліотеки `opengl32.dll`, а ще 52 функції входять до складу бібліотеки утиліт `glu32.dll`.

Базові функції OpenGL забезпечують можливість побудови зображень графічних примітивів, таких як точки, лінії, багатокутники та растрові зображення. Вони також включають функції для перетворення координат, обмеження області видимості, керування кольором, освітленням, текстурою та туманом.

Функції бібліотеки утиліт OpenGL є розширенням базового набору функцій і служать для створення зображень складніших об'єктів, таких як сфери, диски, конічні циліндри. Вони також дозволяють керувати текстурою та виконувати перетворення координат, проводити тріангуляцію багатокутників та побудову кривих та поверхонь на нерегулярній сітці контрольних точок з використанням форм Без'є та раціональних B-сплайнів.

Всі базові функції OpenGL можна розділити на п'ять категорій:

- Функції опису примітивів визначають нижній рівень ієрархії об'єктів, які можуть бути відображені графічною системою. У OpenGL такими примітивами є точки, лінії, багатокутники і т.д.
- Функції опису джерел світла використовуються для визначення положення та параметрів джерел світла, які знаходяться у тривимірній сцені.
- Функції завдання атрибутів дозволяють програмісту визначати зовнішній вигляд відображуваних об'єктів. Ці атрибути включають колір, характеристики матеріалу, текстури, параметри освітлення.
- Функції візуалізації дозволяють задавати положення спостерігача у віртуальному просторі та параметри об'єктива камери. Це

дозволяє системі правильно побудувати зображення і відсікти об'єкти, які не потрапляють в поле зору.

Набір функцій геометричних перетворень дозволяє програмісту здійснювати різноманітні трансформації об'єктів, такі як повороти, зсуви, масштабування.

OpenGL складається з набору бібліотек, і основні функції знаходяться в основній бібліотеці, яку позначають аббревіатурою GL. Крім основної бібліотеки, OpenGL включає кілька додаткових бібліотек. Перша з них - бібліотека утиліт GL (GLU - GL Utility). Усі функції цієї бібліотеки визначаються за допомогою базових функцій GL. Бібліотека GLU містить реалізацію складніших функцій, таких як набір популярних геометричних примітивів (куб, куля, циліндр, диск), функції побудови сплайнів та додаткові операції над матрицями.

OpenGL сам по собі не включає спеціальних команд для роботи з вікнами або отримання інформації від користувача. Тому були створені спеціальні бібліотеки, які забезпечують такі функції, які часто використовуються при взаємодії з користувачем та для відображення інформації за допомогою віконної підсистеми. Однією з найпопулярніших таких бібліотек є GLUT (GL Utility Toolkit). Формально GLUT не є частиною OpenGL, але практично включається в багато дистрибутивів OpenGL і має реалізації для різних платформ. GLUT надає мінімальний набір функцій, необхідних для створення OpenGL-додатків. Існує також менш популярна бібліотека GLX, яка має аналогічні функції.

Крім того, функції, специфічні для конкретної віконної підсистеми, зазвичай входять до її прикладного програмного інтерфейсу (API). Наприклад, функції, специфічні для виконання OpenGL, можуть бути включені в Win32 API для системи Windows або в X Window для системи X Window.

На схемі 3.7, яка представлена нижче, зображена організація системи бібліотек у версії, що працює під управлінням системи Windows.

На схемі видно, що основна бібліотека GL містить базові функції OpenGL. Бібліотека GLU є додатковою і включає більш складні функції. Бібліотека GLUT, хоча не є частиною OpenGL, забезпечує мінімальний набір функцій для створення OpenGL-застосувань. Інші функції, пов'язані з віконною підсистемою, можуть бути включені в Win32 API.

Отже, OpenGL складається з основної бібліотеки GL, додаткової бібліотеки GLU і, за необхідності, використовується разом з бібліотекою GLUT або іншими віконними підсистемами для роботи з вікнами та отримання інформації від користувача.

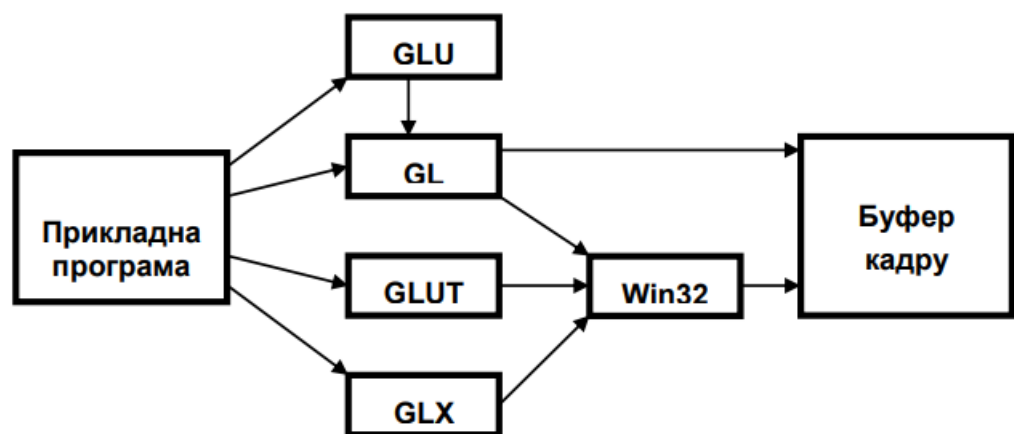


Рис. 3.7. Організація бібліотеки OpenGL

OpenGL використовує модель клієнт-сервер для реалізації своїх функцій. У цій моделі прикладна програма виступає в ролі клієнта, що генерує команди, а сервер OpenGL інтерпретує та виконує ці команди. Сервер може бути розташований на тому ж комп'ютері, що й клієнт (наприклад, у вигляді динамічної бібліотеки - DLL), або на іншому комп'ютері, використовуючи спеціальний протокол передачі даних між машинами.

OpenGL обробляє та формує зображення графічних примітивів в буфері кадру з урахуванням вибраних режимів. Примітив може бути точкою, відрізком, багатокутником тощо. Кожен режим може бути змінений незалежно від інших. Визначення примітивів, вибір режимів та інші операції виконуються за допомогою команд, які викликаються у вигляді функцій прикладної бібліотеки.

Примітиви визначаються набором однієї чи декількох вершин (vertex). Кожна вершина визначає точку, кінець відрізка або кут багатокутника. Кожній вершині приписуються певні дані, такі як координати, колір, нормалі, текстурні координати і т.д., що називаються атрибутами. У більшості випадків кожна вершина обробляється незалежно від інших.

Архітектура OpenGL реалізує конвеєрну схему обробки графічних даних, яка складається з кількох послідовних етапів обробки. На рисунку 3.8 показана ця схема.

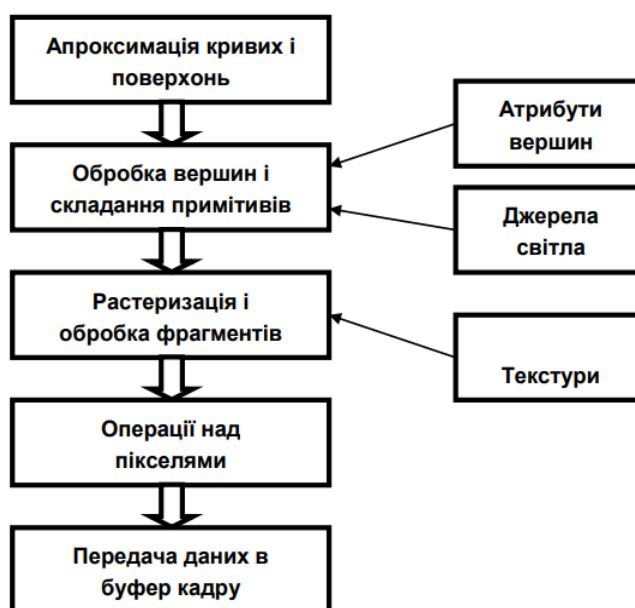


Рис. 3.8. Схема функціонування конвеєра OpenGL

Команди OpenGL завжди обробляються в порядку їх надходження, але можуть відбуватися затримки, перед тим як проявиться ефект від їх виконання. Зазвичай OpenGL реалізує прямий інтерфейс, що означає, що визначення об'єкта призводить до його візуалізації в буфері кадру. Для розробників OpenGL - це набір команд, які керують використанням графічного апарату.

Якщо графічний апарат складається тільки з адресованого буфера кадру, то функції OpenGL повністю реалізуються за допомогою ресурсів центрального процесора. Зазвичай графічний апарат надає різні рівні прискорення, починаючи від апаратної реалізації виводу ліній та багатокутників до високопродуктивних графічних процесорів з підтримкою різних операцій над геометричними даними.

OpenGL є прошарком між апаратним рівнем та рівнем користувача, що дозволяє забезпечити єдиний інтерфейс на різних платформах, використовуючи можливості апаратної підтримки.

3.3. Розробка інтерфейсу

Дуже важливу роль в ефективності роботи програми грає правильно розроблений інтерфейс. Саме від цього буде залежати виконання декількох з основних вимог - зручність і простота в освоєнні. Складний, перевантажений інтерфейс може зменшити ефективність роботи з препроцесором. Головне правило, від якого варто відштовхуватися - інтерфейс не повинен заважати роботі користувача й не повинен відволікати його увагу від роботи, одночасно не втрачаючи при цьому функціональності.

Виходячи з цього, було вирішено використовувати типові для windows-програм візуальні компоненти. Це дасть можливість більшості користувачів швидко розібратися й включитися в роботу. Інтерфейс прямо залежить від функцій, що виконуються програмою.

Інтерфейс програми представлений в вигляді поєднання двох вікон (рис.3.9). Одне з них являється графічним вікном, де створюється модель і редагується. Інше містить в собі основне меню програми та набір команд для роботи з фігурами.

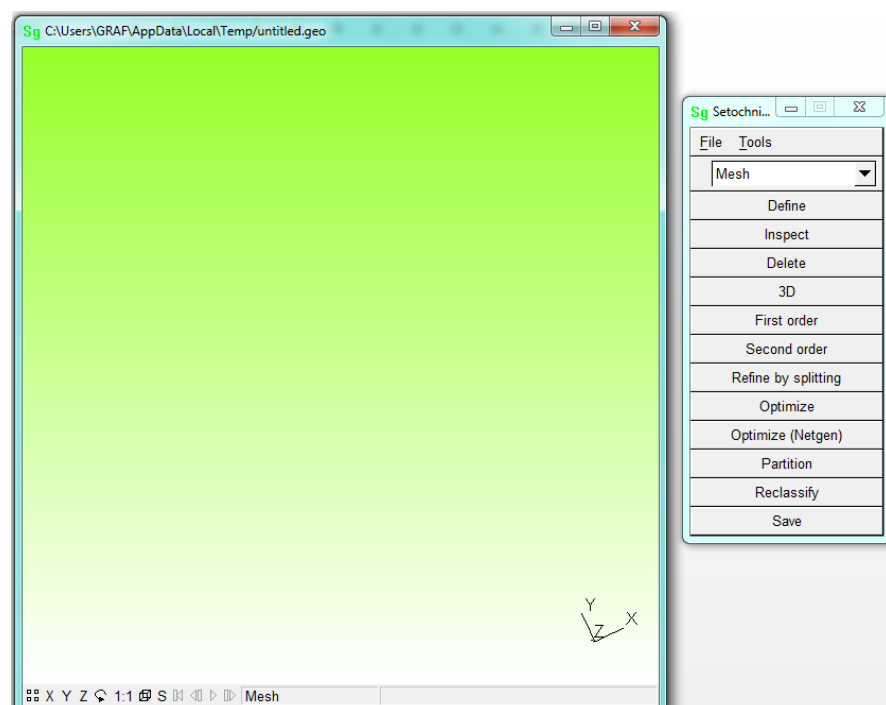


Рис. 3.9. Інтерфейс програмного модулю

В нижній частині Графічного Вікна знаходиться Рядок Стану (Рис.3.10).

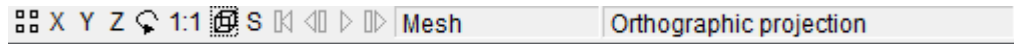


Рис. 3.10. Рядок Стану

Він містить команди для керування зображенням і інформацію про дію активної команди (Табл.3.1).

Таблиця 3.1

Команди Рядка Стану

Зображення піктограми	Комбінація клавіш	Дія
	Alt+x, Alt+Shift+x	Вид моделі відносно осі X.
	Alt+y, Alt+Shift+y	Вид моделі відносно осі Y.
	Alt+z, Alt+Shift+z	Вид моделі відносно осі Z.
	Shift Alt	Поворот моделі на 90° за часовою стрілкою та проти часової. Синхронізація повороту.
	Alt	Встановлення масштабу. Синхронізація масштабу.
	Alt+o, Alt+Shift+o	Перемикання режиму проекції.
	Esc	Перемикач миші Вкл/Викл.

Опис змісту діалогу меню File

Відкриття фігури

1. Натискаємо на меню **File**.
2. З випадаючого меню обираємо **Open** або користуємося комбінацією клавіш **Ctrl+O**.
3. В списку знаходимо папку, в якій зберігаються фігури.
4. Подвійним натисканням відкриваємо папку.
5. Подвійним натисканням обираємо фігуру

Об'єднання декількох елементів фігур

1. Натискаємо на меню File.
2. З випадаючого меню обираємо **Merge, Shift+Ctrl+O**.

3. Вибираємо спочатку одну фігуру, а потім таким чином обираємо ще одну .

Відкриття нового вікна

1. Натискаємо на меню File.
2. З випадаючого меню обираємо New Window.

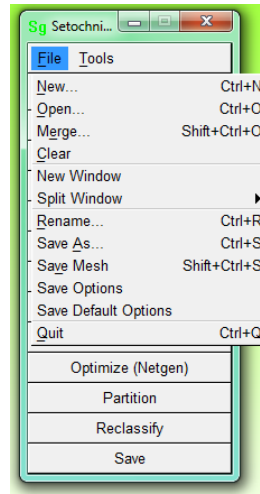


Рис. 3.11. Меню File

Розділення вікна

1. Натискаємо на вікно File.
2. З випадаючого вікна обираємо Split Window => Horizontally для горизонтального розділення, Split Window =>Vertically для вертикального розділення.

Збереження проекту

1. Натискаємо на меню **File**.
2. З випадаючого меню обираємо **Save As** або **Ctrl+S**.
3. Вибираємо папку, в яку виконається зберігання .
4. В поле **Ім'я файлу** задаємо ім'я моделі.
5. Натиснути кнопку Сохранить.

Закриття проекту

1. Натискаємо на меню File.
2. З випадаючого меню обираємо Quit.

Опис змісту діалогу меню Tools

Опції налаштування відображення

1. Натискаємо на меню **Tools**.
2. З випадаючого меню обираємо Options.
3. З'являється вікно Options General (Рис. 3.12).

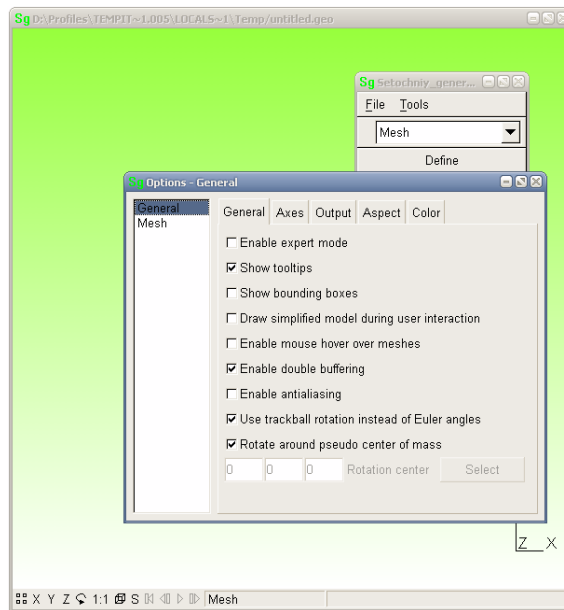


Рис. 3.12. Вікно Options General

4. При виборі Mesh на вкладці та Visibility можливо обирати відображення на геометричній фігурі точок, ліній, поверхонь та об'ємів та нумерацію точок, ліній, поверхонь та об'ємів (рис.3.13).

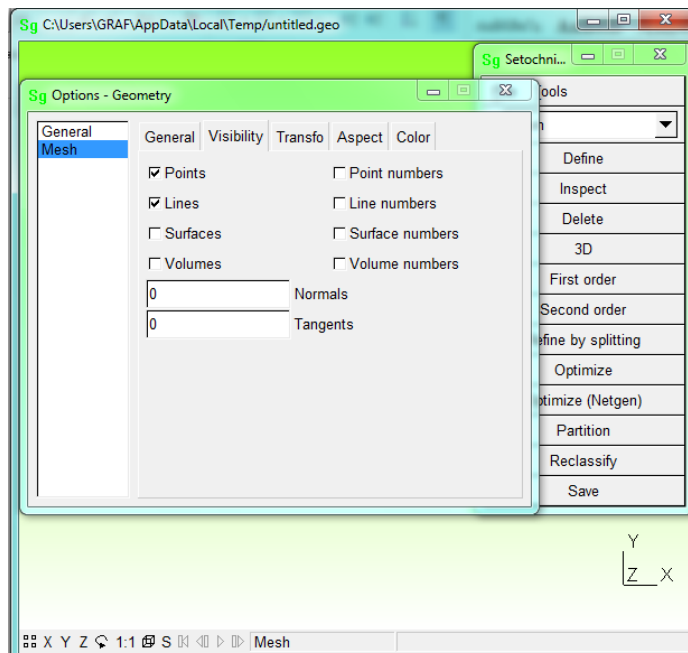


Рис. 3.13. Вікно Options Mesh

Опції Statistics

1. Натискаємо на меню **Tools**.

2. З випадального меню обираємо *Statistics*.
3. З'являється вікно *Statistics* (Рис. 3.14).

Показує інформацію про кількість елементів геометрії, сітки, а також деякі дані про якість сітки.

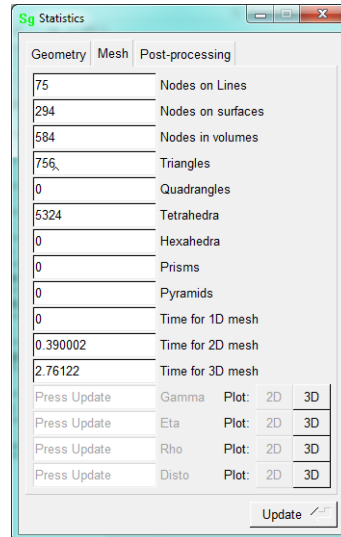


Рис. 3.14. Вікно *Statistics*

Gamma = (наближено) радіус вписаного сфери/радіус описаної сфери.

Eta = (наближено) обсяг тетраедра в ступені $(2/3)$ / сума довжин ребер в ступені 2.

Rho = (наближено) найменша довжина ребра/найбільша довжина ребра.

Опції *Clipping*

1. Натискаємо на меню **Tools**.
2. З випадального меню обираємо *Clipping*.
3. З'являється вікно *Clipping*.

Перетин - напевно, ще більш важливий інструмент перегляду сітки, ніж *Visibility*. Адже як інакше заглянути всередину моделі, якщо не зробити розтин? Gmsh дозволяє робити розтин геометрії і сітки. Інтерфейс наступний:

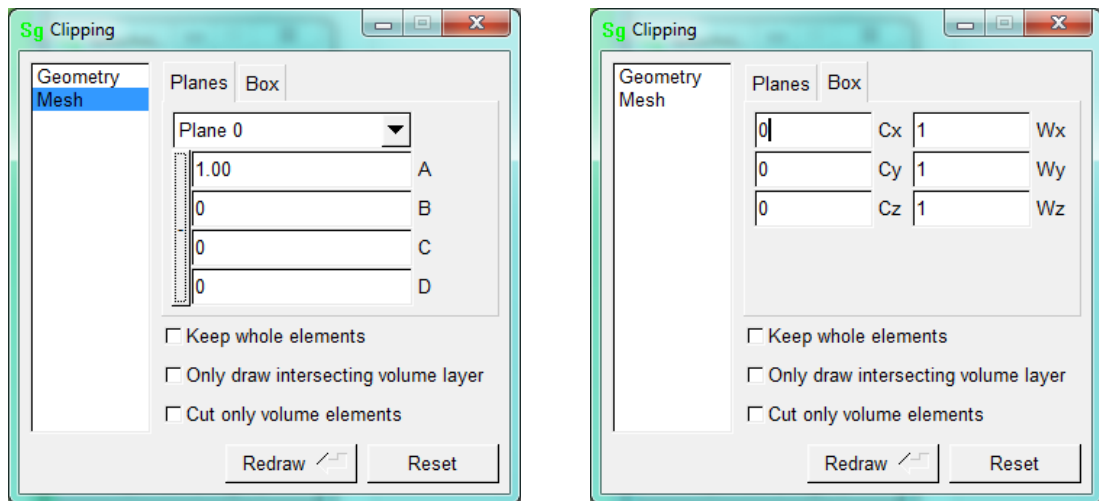


Рис. 3.15. Вікно Clipping

І для геометрії і для сітки можна зробити перетин або площинами (Planes) або паралелепіпедом (Box). Для цього потрібно вибрати відповідну вкладку.

Planes. Можна задати 6 січних площин коефіцієнтами A, B, C, D з рівняння площині $A \cdot x + B \cdot y + C \cdot z + D = 0$. Можливо також інтерактивний "рух" площин за допомогою миші. Для цього порухайте мишею з затиснутою лівою кнопкою в полі введення числа для коефіцієнта. Ви побачите, як змінюється площину перетину і, відповідно, саме перетин сітки.

Box. Cx, Cy, Cz - координати центру перетину паралелепіпеда. Wx, Wy, Wz - довжини відповідних сторін, які симетричні центру. Таким чином, задані за замовчуванням значення $(Cx, Cy, Cz) = (0, 0, 0)$ і $(Wx, Wy, Wz) = (1, 1, 1)$ визначають перетин паралелепіпеда в межах від $(-0.5, -0.5, -0.5)$ до $(0.5, 0.5, 0.5)$.

Приблизно такий перетин сітки площиною ви отримаєте, визначивши коефіцієнт D:

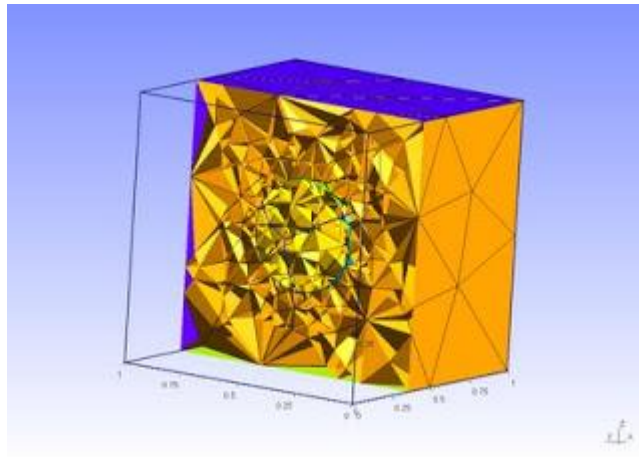


Рис. 3.16. Перетин кубу

Якщо ви поставите галочку навпроти *Keep whole elements* (зберегти цілі елементи), то отримаєте ще більш цікаву картину, дивлячись на яку вже можна говорити про якість тетраедрів (звичайно, чисто візуально), про згущення сітки і т.д.

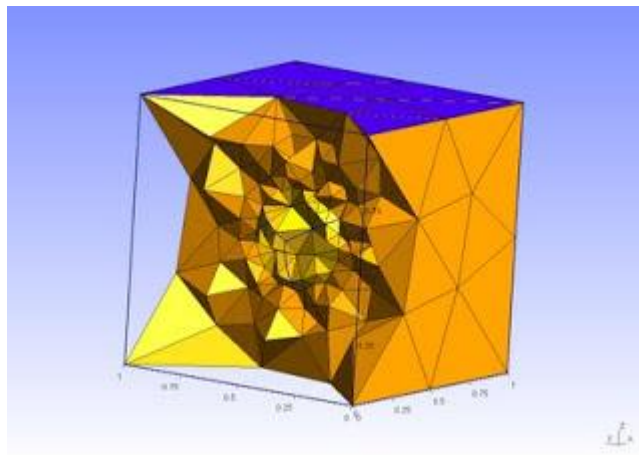


Рис. 3.17. Перетин кубу

3.4. Генерація сітки в програмному додатку

Для того щоб побудувати тривимірну сітку спочатку треба відкрити файл з геометрією тіла (приклад рис. 3.18).

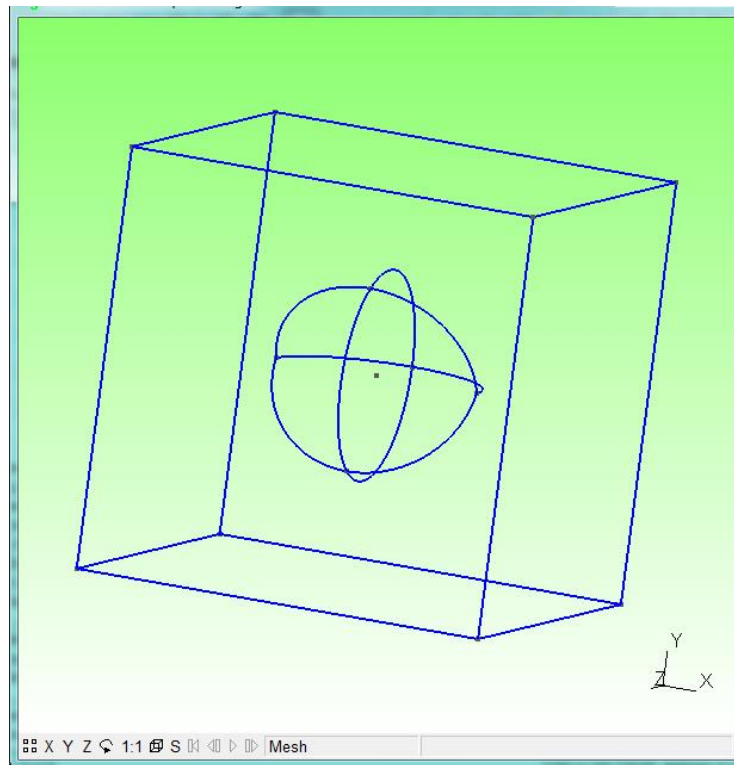


Рис. 3.18. Тривимірне складне тіло

Обираємо **3D** і автоматично програмний додаток будує тривимірну сітку (автоматично викликає спочатку 2D).

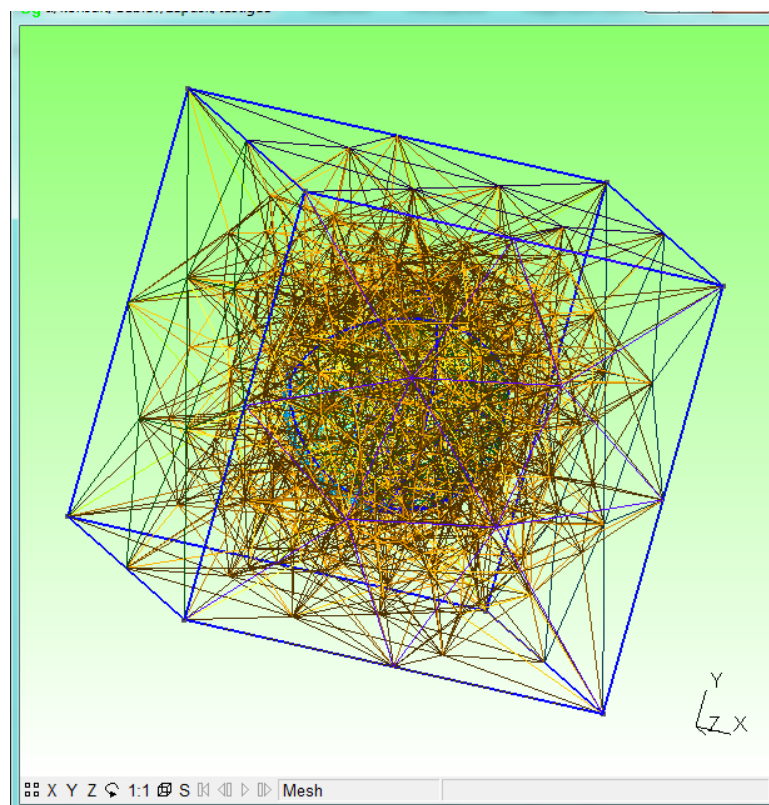


Рис. 3.19. Тривимірна сітка

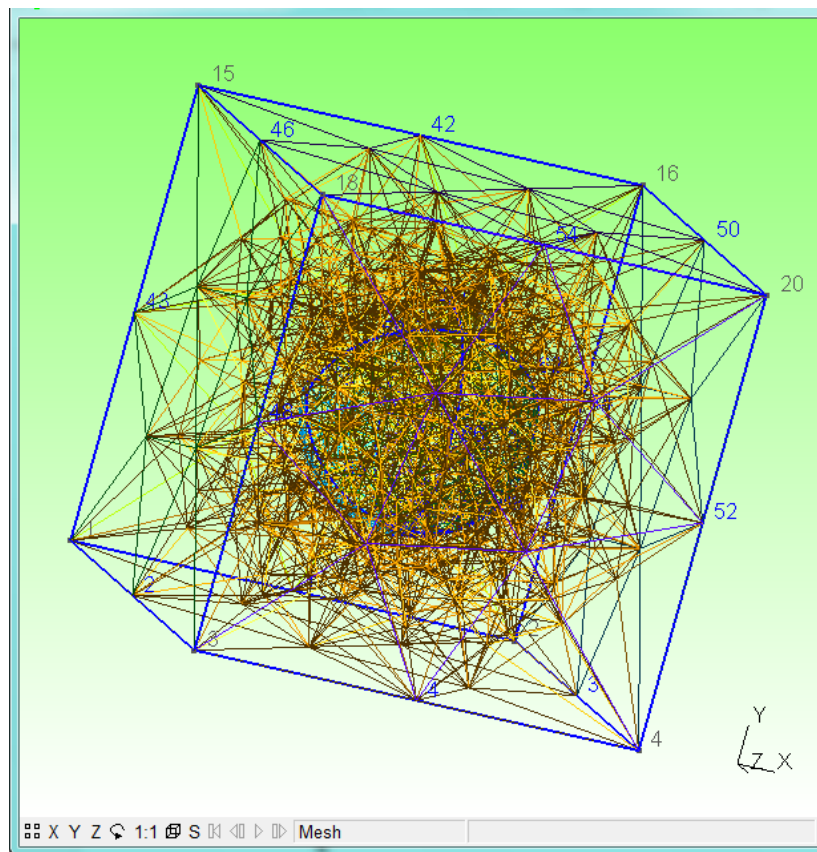


Рис. 3.20. Тривимірна сітка з нумерацією точок

Якщо оберемо **Second order**, то буде побудова сітки з елементами другого порядку (є можливість побудови сіток більш високого порядку - 3-го, 4-го і 5-го, проте робиться це через. Geo файл.

Якщо оберемо **Refine by splitting**, то буде подрібнення за рахунок розбиття - наявні елементи сітки б'ються всередині себе для отримання більш дрібної сітки.

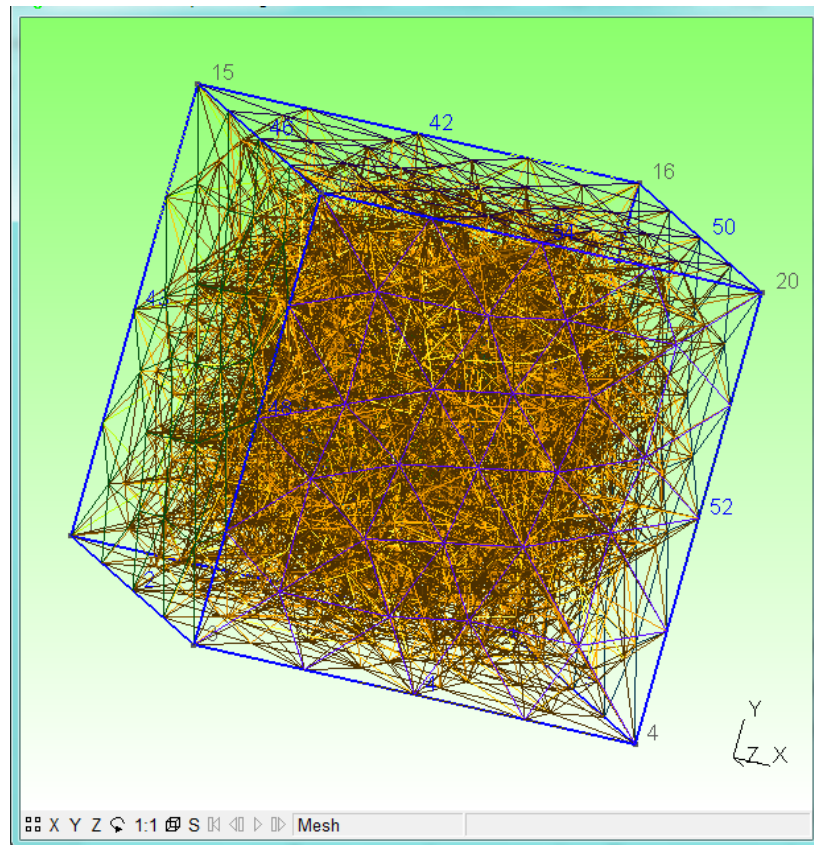


Рис. 3.21. Більш дрібна тривимірна сітка

Розглянемо опції програмного додатку які принципово впливають на побудову сітки, якість її елементів, а також їх збереження. У дужках після назви опції стоїть значення за замовчуванням.

Geometry.ExtrudeReturnLateralEntities (=1) - чи потрібно додавати додаткові номери в масив, що повертається командою Extrude.

Geometry.ExtrudeSplinePoints (=5) - кількість контрольних точок для сплайнів, створених внаслідок витягування.

Geometry.NumSubEdges (=20) - кількість відрізків між контрольними точками при відображенні кривих (ця опція не стосується побудови сітки, проте дозволяє сильно поліпшити відображення складних кривих).

Geometry.OCCFixSmallEdges (=1) - фіксувати малі ребра в IGES, STEP і BRep моделях (важлива опція при експорті моделей з інших CAD програм).

Geometry.OCCFixSmallFaces (=1) - фіксувати малі грані в IGES, STEP і BRep моделях (важлива опція при експорті моделей з інших CAD програм).

Geometry.Tolerance (=1e-06) - геометрична точність (допустиме відхилення).

Mesh.Algorithm3D (=1 - алгоритм побудови тетраедральної сітки (1-Delaunay, 2-Frontal). Алгоритм Delaunay найбільш стійкий і швидкий, однак іноді він змінює поверхневу сітку (спочатку будує одновимірну сітку, потім засновану на ній двовимірну, а потім засновану на двовимірній тривимірну сітку) і не підходить для створення змішаної/структурованої/неструктурованою сітки. Для цих випадків підходить Frontal алгоритм. Якість елементів, вироблених цими алгоритмами, приблизно однакове.

Mesh.Binary (=0) - записувати сітку в бінарному форматі.

Mesh.CharacteristicLengthFactor (=1) - множник характеристичної довжини (застосовується, щоб подрібнити/згрубіть всі елементи сітки в рівній мірі).

Mesh.CharacteristicLengthMin (=0) - мінімальний розмір елемента сітки.

Mesh.CharacteristicLengthMax (=1e+22) - максимальний розмір елемента сітки.

Mesh.CpuTime (=0) - час (у секундах), відведений для побудови сітки.

Mesh.ElementOrder (=1) - порядок елементів сітки (1 - лінійні елементи; доступні також 2, 3, 4 і 5 порядки).

Mesh.MinimumCirclePoints (=7) - мінімальна кількість точок для апроксимації окружності.

Mesh.MinimumCurvePoints (=3) - мінімальна кількість точок для апроксимації кривої лінії.

Mesh.NumSubEdges (=2) - кількість ребер розбиття при відображенні елементів високих порядків. Ця опція важлива тільки з точки зору візуалізації. Проілюструємо це на прикладі розбиття кола грубою сіткою другого порядку (зліва - кількість ребер = 2, праворуч - кількість ребер = 5):

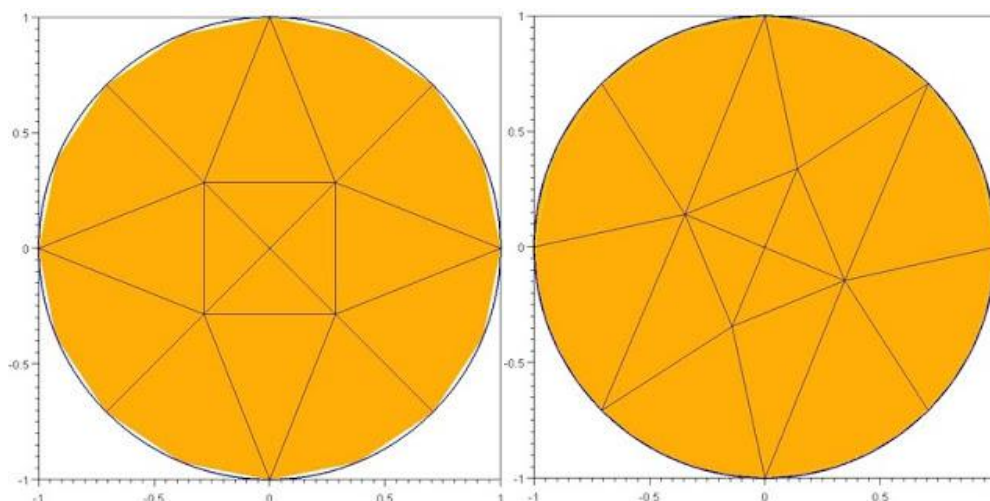


Рис. 3.22. Розбиття кола грубою сіткою

Mesh.SaveAll (=0) - зберігати всі елементи сітки, незалежно від фізичних об'єктів.

Mesh.SecondOrderIncomplete (=0) - створювати елементи неповного другого порядку (8 - вузлові чотирикутники, 20 - вузлові шестигранники і т.д.).

Mesh.SecondOrderLinear (=0) - чи повинні вершини елементів другого порядку бути отримані лінійною інтерполяцією (у цьому випадку, додаткові вузли ставляться строго на середину ребра елемента. Інакше, ребра ламаються на два, а додаткові вузли зміщуються для більш точної апроксимації). Проілюструємо це (ліворуч опція відключена, праворуч - включена):

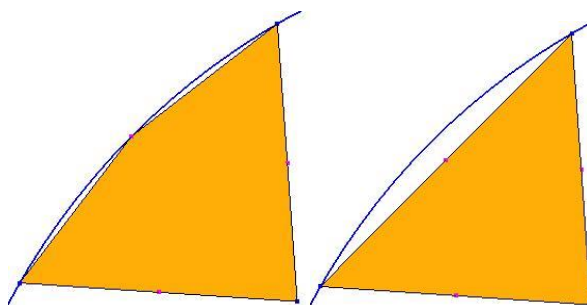


Рис. 3.23. Лінійна інтерполяція

3.5. Оптимізація сітки в програмному додатку

Optimize і Optimize (Netgen) - оптимізація сітки, для отримання більш якісних елементів, по "своєму" алгоритмом і за алгоритмом Netgen'a відповідно.

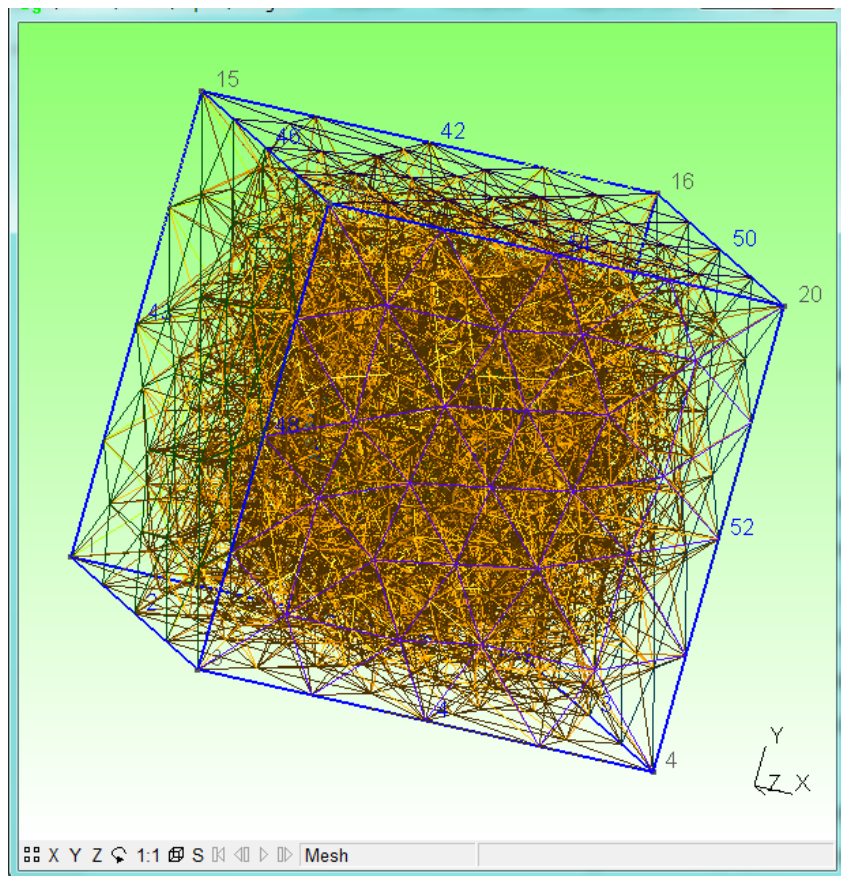


Рис. 3.24. Оптимізація сітки

Зупинимося на опціях, які принципово впливають на якість сітки, її елементів, а також їх збереження. У дужках після назви опції стоїть значення за замовчуванням.

Mesh.Optimize (=0) - оптимізувати сітку для підвищення якості тетраедральних елементів.

Mesh.OptimizeNetgen (=0) - оптимізувати сітку для підвищення якості тетраедральних елементів, використовуючи алгоритм Netgen'a.

Mesh.RecombinationAlgorithm (= 0) - алгоритм рекомбінування (0 - стандартний, 1 - blossom).

Mesh.Smoothing (=1) - кількість кроків згладжування, що застосовується до підсумкової сітці.

3.6. Формати файлів в програмному додатку

Програмний додаток нічого сам не обчислює, тому він має бути пов'язаний з іншими програмами за допомогою створюваних і прочитуваних ним файлів.

Зробити це можна через випадне меню "File" Вікна Меню, вибравши "Open" або "Save As" (рис. 3.25).

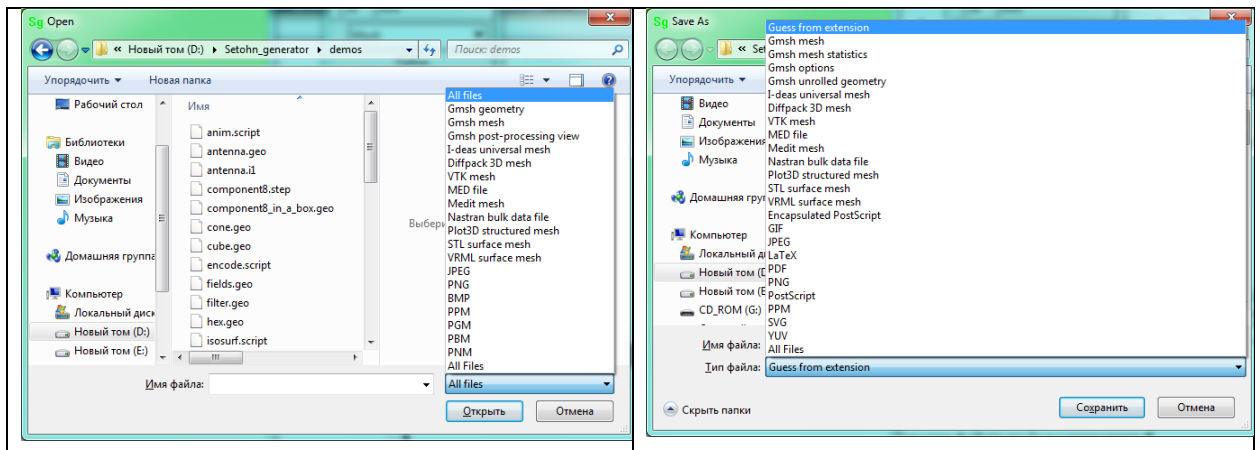


Рис. 3.25. Випадне меню "File": "Open" та "Save As"

Розглянемо деякі формати файлів по їх розширеннях:

- Gmsh geometry .geo - файл геометрії GMSH.
- STEP і IGES - відомі формати файлів геометрія, підтримувана багатьма CAD -пакетами. Імпорт STEP і IGES, розроблений на основі відкритої бібліотеки OpenCascade дозволяє завантажувати в програму моделі, побудовані в сторонніх CAD -ах (зворотне вивантаження моделі з .geo в ці формати доки не передбачена), в т.ч. у відкритому пакеті Salome.
- STL surface mesh - поширений формат для передачі поверхонь через трикутні сітки.
- .msh файл - файл сітки генератора.

Msh-файл складається з секції з інформацією про формат файлу (MeshFormat) і секцій. Дана інформація описує вузли (Nodes) й елементи (Elements) сітки:

```
$MeshFormat
$EndMeshFormat
$Nodes
nNodes
iNode x y z
...
$EndNodes
$Elements
nElements
iElement type nTags tags nodes
...
$EndElements
```

В даному прикладі:

nNodes – кількість вузлів сітки,

i – індекс вузла,

x, y, z – координати вузла,

nElements – кількість елементів,

type – геометричний тип елементу (таблиця. 3.2),

nTags – кількість тегів,

tags – список тегів,

nodes – список вузлів з яких складається елемент.

За замовчуванням, у msh-файлі задається три теги: перший вказує номер фізичної сутності, до якої даний елемент належить, другий геометричну область, третє - номер частини сітки, до якої належить елемент.

Таблиця 3.2

Геометричні типи елементів

Номер	Геометричний тип
1	Лінія
2	Трикутник
3	Чотирикутник
4	Тетраедр
5	Гексаедр

3.7. Висновки до розділу

Розроблено загальний алгоритм роботи програми й інтерфейс на основі передбачуваної функціональності, були обрані програмні засоби для реалізації поставленого завдання. Програмний додаток реалізовано на мові програмування C++ в середовищі Visual C++ 2022. При створенні засобів відображення і візуалізації використаний графічний інтерфейс OpenGL.

У програмному додатку реалізовано модуль сітка. У модулі сітка доступно створення 3-х вимірних сіток і їх оптимізація.

ВИСНОВКИ

У магістерській роботі було розглянуто аналіз та оптимізацію алгоритмів побудови дискретного представлення складних об'єктів.

У роботі запропоновано технологію автоматичної побудови тетраедральних сіток для складних областей на основі методу рухомого фронту. Проведено аналіз впливу обчислювальних похибок на алгоритми при їх реалізації на ЕОМ, представлено теоретичне обґрунтування кінцівки роботи запропонованих алгоритмів.

Для досягнення поставленої мети були вирішені наступні завдання:

- Проаналізовано основні поширені способи подання геометричних тіл та методи й алгоритми побудови сіток просторових областей.
- Розглянуто та проаналізовано критерії якості побудованої сітки.
- Розроблено і реалізовано алгоритми побудови конформних тетраедральних сіток, узгоджених із заданою дискретною межею, для багатокomпонентних та багатогранних областей.
- Розроблено додаток для тривимірної тріангуляції просторових конструкцій. Програма реалізована на мові програмування C++ в середовищі Visual C++ 2022. При створенні засобів відображення і візуалізації використаний графічний інтерфейс OpenGL.

Практичною цінністю роботи є розробка додатку автоматичної генерації тривимірних розрахункових сіток для скінчено – елементного моделювання конструкцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities // <http://gmsh.info>
2. Arantes, E.R., Oliveira, E., Babuska, I., Zienkiewicz, O.C., Gado J.P. Proceedings International Conference on Accuracy Estimates and Adaptive Refinement in Finite Element Computation, Lisbon, 1984.
3. Baldwin K.H., Schreyer H.L. Automatic generation of quadrilateral elements by a conformal mapping. Eng. Comput. 1985, V.2, p. 187-194.
4. Blum H A transformation for extracting new descriptors of shape. In Models for the perception of speech and visual formed. By W.Wathen-Dunn, Cambridge, MA, the MIT Press, 1967, p.326-380
5. Brawn P.R. A non-interactive method for the automatic generation of finite element meshes using Schwarz-Christoffel transformation. Comp. Methods in Appl. Mech. Eng., 1981, V. 25, p. 101-126.
6. Cook W.A. Body oriented (natural) coordinates for generating three-dimensional meshes. Int. J. Num. Meth. Eng., 1974, V.8, p. 27-43.
7. D.A. Field The legacy of automatic mesh generation from solid modeling, Computer-Aided Geometric Des,V12, 1995,651-674
8. Fleischmann P. Mesh Generation for Technology CAD in Three Dimensions [Электронный ресурс] / P. Fleischmann // Dissertation. - 1999. - Режим доступа: <http://www.iue.tuwien.ac.at/phd/fleischmann/diss.html>.
9. Foley J, A.van Dam, S.Feiner, J.Hughes, R.Philips Introduction to Computer Graphics, - Addison Wesley, 1994
- 10.Fomenko A.T., Kunii T.L. Topological modeling for visualization, Springer-Verlag, Tokyo and Heidelberg, 1997
- 11.Frey, P.J., George, P.-L. Mesh Generation: Application to Finite Elements - HERMES Science Europe, OXFORD & PARIS, 2000, 814p.
- 12.Fritsch F, Piccinini R.A. Cellular structures in topology. Cambridge University Press, Cambridge, 1990

13. George P.-L., Borouchaki H., Saltel E. 'Ultimate' robustness in meshing an arbitrary polyhedron // Int. J. Numer. Meth. Eng. 2003. Vol. 58. Pp. 1061–1089.
14. Geuzaine C., Remacle J.F. Gmsh: a three-dimensional finite element mesh generator with builtin pre- and post-processing facilities // International Journal for Numerical Methods in Engineering, 2009, Vol. 79, No 11, pp. 1309–1331.
15. Ho-Le, K Finite. Element mesh generation methods: a review and classification. / CAD, 1988, V.20, N 1, p. 27-38.
16. I. Babushka, W.C. Rheinboldt. A-posteriori Error Estimates for Finite Element Method // Int. J. Numer. Meth. Eng., Vol. 12, p.p. 1597-1615, 1978.
17. Marot C., Pellerin J., Remacle J.F. One machine, one minute, three billion tetrahedral // International Journal for Numerical Methods in Engineering, 2019, Vol. 117, No 9, pp. 967–990.
18. Schöberl J. NETGEN. An advancing front 2d/3d mesh generator based on abstract rules // Computing and Visualization in Science, 1997, Vol. 1, No 1, pp. 41–52. 11. Ermakov M.K., Kryukov I.A. Supercomputer modeling of flow past hypersonic flight vehicles // Journal of Physics: Conference Series, 2017, Vol. 815, 012016
19. Si H. TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, 2006 // www.wias-berlin.de/software/tetgen/files/tetgenmanual.pdf
20. Si H. TetGen. A delaunay-based tetrahedral mesh generator // ACM Transactions on Mathematical Software, 2015, Vol. 41, No 2, Article 11, pp. 1–36.
21. Александров П.С. Комбинаторная топология. ОГИЗ, Гостехиздат, 1947
22. Галанин М.П. Разработка и реализация алгоритмов трехмерной триангуляции сложных пространственных областей: итерационные методы / Галанин М.П., Щеглов И.А. - М.: ИПМ им. М.В. Келдыша РАН,

2006. – № 9. – 32 с. - (Препринт / РАН, ИПМ им. М.В. Келдыша; 06-01-00421).
23. Галанин М.П. Разработка и реализация алгоритмов трехмерной триангуляции сложных пространственных областей: прямые методы / М.П. Галанин, И.А. Щеглов. – М.: ИПМ им. М.В. Келдыша РАН, 2006. – №10. – 32 с. – (Препринт / РАН, ИПМ им. М.В. Келдыша; 06-01-00421).
24. Голованов Н.Н. Геометрическое моделирование / Н.Н. Голованов. – М.: Издво Физ.-мат. лит., 2002. – 472 с.
25. Дижевский, А. Ю. Общий подход к реализации методов построения триангуляции неявно заданных поверхностей, использующих разбиение пространства на ячейки / А. Ю. Дижевский // Вычислительные методы и программирование. – 2007. – Т. 8. – С. 286–296.
26. Карташева Е.Л. Инструментальные средства подготовки и анализа данных для решения трехмерных задач математической физики - Математическое моделирование. 1997. Т. 9. №6. С. 20.
27. Ласло М. Вычислительная геометрия и компьютерная графика на C++ / Ласло М. Пер. с англ. М.: БИНОМ, 1997. 304 с.
28. Математическое моделирование: Проблемы и результаты, -М. Наука, 2003
29. Новые возможности Visual Studio 2022 [Электронный ресурс] // Microsoft Learn: [web-сайт]. – Режим доступа: <https://learn.microsoft.com/ruru/visualstudio/ide/whats-new-visual-studio-2022?view=vs-2022>
30. Пасько А.А., Пилюгин В.В., Покровский В.Н. Геометрическое моделирование в задаче анализа функций трех переменных, Сообщение ОИЯИ Р10-86-310, Дубна, 1986. Publication in English: Computers and Graphics, vol.12, # 3/4, 1988, pp. 457-465.
31. Построение расчетных сеток: Теория и приложения – В сб. под ред. Иваненко С.А., Гаранжа В.А., ВЦ РАН, М. 2002

- 32.Препарата Ф. Вычислительная геометрия: Введение /Препарата Ф., Шеймос М. Пер. с англ. М.: Мир, 1989. 478 с.
- 33.Рвачев Л. Методы логической алгебры в математической физике. Наукова думка, Киев, 1974г.
- 34.Роджерс Д. Математические основы машинной графики / Роджерс Д., Адаме Дж. Пер. с англ. М.: Машиностроение, 1980. 204 с.
- 35.Скворцов А.В. Обзор алгоритмов построения триангуляции Делоне / А.В. Скворцов // Вычислительные методы и программирование. – 2002. – Т.3. – С. 14-39.
- 36.Скворцов А.В. Применение триангуляции для решения задач вычислительной геометрии / Скворцов А.В., Костюк Ю.Л. Геоинформатика: Теория и практика. Вып. 1. Томск: Изд-во Томск, ун-та, 1998. С. 127-138.
- 37.Скворцов А.В. Триангуляция Делоне и ее применение / А.В. Скворцов // Томск, Изд-во Том. ун-та, 2002. С. 128.
- 38.Соллогуб А.И., Вальшин А.Т. Система трехмерного геометрического моделирования пространственных тел с использованием характеристических и R-функций. Программирование, 1991, N3, с.86-96
- 39.Сьярле Ф. Метод конечных элементов для эллиптических задач / Ф. Сьярле; пер. с англ. Б.И. Квасова. – М.: Изд-во «Мир», 1980. – 512 с.
- 40.Химушин Ф.Ф. Конструктивное геометрическое моделирование. –В. сб. Программное обеспечение САПР. ВЦАН СССР, М., 1987, с.102-121
- 41.Химушин Ф.Ф. Обзор методов геометрического моделирования для САПР.- В сб. Программное обеспечение САПР, ВЦАН СССР, М., 1987, с.78-101
- 42.ЧельцовА.Ю., ДавыдченкоЭ.А. Представление данных и алгоритмы для системы геометрического моделирования, основанной на заметании. - В сб. Программное обеспечение САПР, ВЦАН СССР, М., 1987, с.121-133.
- 43.Шайдуров В.В. Многосеточные методы конечных элементов / В.В. Шайдуров. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 288 с.

44.Шайдуров В.В. Многосеточные методы конечных элементов. - М.,
Наука, 1989. - 288с.

ДОДАТОК А

Лістинг коду

Generator.h

```
#ifndef _GENERATOR_H_
#define _GENERATOR_H_
class GModel;
void GetStatistics(double stat[50], double quality[4][100]=0);
void AdaptMesh(GModel *m);
void GenerateMesh(GModel *m, int dimension);
void OptimizeMesh(GModel *m);
void OptimizeMeshNetgen(GModel *m);
void RefineMesh(GModel *m, bool linear, bool splitIntoQuads=false,
               bool splitIntoHexas=false);
#endif
```

Generator.cpp

```
#include <stdlib.h>
#include "GmshConfig.h"
#include "GmshMessage.h"
#include "Numeric.h"
#include "Context.h"
#include "OS.h"
#include "GModel.h"
#include "MLine.h"
#include "MTriangle.h"
#include "MQuadrangle.h"
#include "MTetrahedron.h"
#include "MHexahedron.h"
#include "MPrism.h"
#include "MPyramid.h"
#include "meshGEdge.h"
#include "meshGFace.h"
#include "meshGFaceBDS.h"
#include "meshGRegion.h"
#include "BackgroundMesh.h"
#include "BoundaryLayers.h"
#include "HighOrder.h"
#include "Generator.h"
#ifdef HAVE_NO_POST
#include "PView.h"
#include "PViewData.h"
#endif
static MVertex* isEquivalentTo(std::multimap<MVertex*, MVertex*>&m, MVertex *v)
{
    std::multimap<MVertex*, MVertex*>::iterator it = m.lower_bound(v);
    std::multimap<MVertex*, MVertex*>::iterator ite = m.upper_bound(v);
    if (it == ite) return v;
    MVertex *res = it->second; ++it;
    while (it != ite){
        res = std::min(res, it->second); ++it;
    }
    if (res < v) return isEquivalentTo(m, res);
    return res;
}
static void buildASetOfEquivalentMeshVertices(GFace *gf,
        std::multimap<MVertex*, MVertex*>&equivalent,
```

```

        std::map<GVertex*, MVertex*>&bm)
{
    std::list<GEdge*> edges = gf->edges();
    std::list<GEdge*> emb_edges = gf->embeddedEdges();
    std::list<GEdge*>::iterator it = edges.begin();
    while(it != edges.end()){
        if((*it)->isMeshDegenerated()){
            MVertex *va = ((*it)->getBeginVertex()->mesh_vertices.begin());
            MVertex *vb = ((*it)->getEndVertex()->mesh_vertices.begin());
            if (va != vb){
                equivalent.insert(std::make_pair(va, vb));
                equivalent.insert(std::make_pair(vb, va));
                bm[(*it)->getBeginVertex()] = va;
                bm[(*it)->getEndVertex()] = vb;
                printf("%d equivalent to %d\n", va->getNum(), vb->getNum());
            }
        }
        ++it;
    }
    it = emb_edges.begin();
    while(it != emb_edges.end()){
        if((*it)->isMeshDegenerated()){
            MVertex *va = ((*it)->getBeginVertex()->mesh_vertices.begin());
            MVertex *vb = ((*it)->getEndVertex()->mesh_vertices.begin());
            if (va != vb){
                equivalent.insert(std::make_pair(va, vb));
                equivalent.insert(std::make_pair(vb, va));
                bm[(*it)->getBeginVertex()] = va;
                bm[(*it)->getEndVertex()] = vb;
            }
        }
        ++it;
    }
}

struct geomTresholdVertexEquivalence
{
    // Initial MVertex associated to one given MVertex
    std::map<GVertex*, MVertex*> backward_map;
    // initiate the forward and backward maps
    geomTresholdVertexEquivalence(GModel *g);
    // restores the initial state
    ~geomTresholdVertexEquivalence ();
};

geomTresholdVertexEquivalence::geomTresholdVertexEquivalence(GModel *g)
{
    std::multimap<MVertex*, MVertex*> equivalenceMap;
    for (GModel::fiter it = g->firstFace(); it != g->lastFace(); ++it)
        buildASetOfEquivalentMeshVertices(*it, equivalenceMap, backward_map);
    // build the structure that identifiates geometrically equivalent
    // mesh vertices.
    for (std::map<GVertex*, MVertex*>::iterator it = backward_map.begin();
        it != backward_map.end(); ++it){
        GVertex *g = it->first;

        MVertex *v = it->second;
        MVertex *other = isEquivalentTo(equivalenceMap, v);
        if (v != other){

```



```

printf("Finally : %d equivalent to %d\n", v->getNum(), other->getNum());
g->mesh_vertices.clear();
g->mesh_vertices.push_back(other);
std::list<GEdge*> ed = g->edges();
for (std::list<GEdge*>::iterator ite = ed.begin() ; ite != ed.end() ; ++ite){
    std::vector<MLine*> newl;
    for (unsigned int i = 0; i < (*ite)->lines.size(); ++i){
        MLine *l = (*ite)->lines[i];
        MVertex *v1 = l->getVertex(0);
        MVertex *v2 = l->getVertex(1);
        if (v1 == v && v2 != other){
            delete l;
            l = new MLine(other,v2);
            newl.push_back(l);
        }
        else if (v1 != other && v2 == v){
            delete l;
            l = new MLine(v1,other);
            newl.push_back(l);
        }
        else if (v1 != v && v2 != v)
            newl.push_back(l);
        else
            delete l;
    }
    (*ite)->lines = newl;
}
}
}
}

```

```

geomTresholdVertexEquivalence::~geomTresholdVertexEquivalence()
{
    // restore the initial data
    for (std::map<GVertex*, MVertex*>::iterator it = backward_map.begin();
        it != backward_map.end() ; ++it){
        GVertex *g = it->first;
        MVertex *v = it->second;
        g->mesh_vertices.clear();
        g->mesh_vertices.push_back(v);
    }
}

```

```

template<class T>
static void GetQualityMeasure(std::vector<T*>&ele,
                             double &gamma, double &gammaMin, double &gammaMax,
                             double &eta, double &etaMin, double &etaMax,
                             double &rho, double &rhoMin, double &rhoMax,
                             double &disto, double &distoMin, double &distoMax,
                             double quality[4][100])
{
    for(unsigned int i = 0; i < ele.size(); i++){
        double g = ele[i]->gammaShapeMeasure();
        gamma += g;
        gammaMin = std::min(gammaMin, g);
        gammaMax = std::max(gammaMax, g);
        double e = ele[i]->etaShapeMeasure();
    }
}

```

```

eta += e;
etaMin = std::min(etaMin, e);

etaMax = std::max(etaMax, e);
double r = ele[i]->rhoShapeMeasure();
rho += r;
rhoMin = std::min(rhoMin, r);
rhoMax = std::max(rhoMax, r);
double d = ele[i]->distoShapeMeasure();
disto += d;
distoMin = std::min(distoMin, d);
distoMax = std::max(distoMax, d);
for(int j = 0; j < 100; j++){
    if(g > j / 100. && g <= (j + 1) / 100.) quality[0][j]++;
    if(e > j / 100. && e <= (j + 1) / 100.) quality[1][j]++;
    if(r > j / 100. && r <= (j + 1) / 100.) quality[2][j]++;
    if(d > j / 100. && d <= (j + 1) / 100.) quality[3][j]++;
}
}
}
void GetStatistics(double stat[50], double quality[4][100])
{
    for(int i = 0; i < 50; i++) stat[i] = 0.;
    GModel *m = GModel::current();
    if(!m) return;
    stat[0] = m->getNumVertices();
    stat[1] = m->getNumEdges();
    stat[2] = m->getNumFaces();
    stat[3] = m->getNumRegions();
    std::map<int, std::vector<GEntity*>> physicals[4];
    m->getPhysicalGroups(physicals);
    stat[45] = physicals[0].size() + physicals[1].size() +
        physicals[2].size() + physicals[3].size();
    for(GModel::eiter it = m->firstEdge(); it != m->lastEdge(); ++it)
        stat[4] += (*it)->mesh_vertices.size();
    for(GModel::fiter it = m->firstFace(); it != m->lastFace(); ++it){
        stat[5] += (*it)->mesh_vertices.size();
        stat[7] += (*it)->triangles.size();
        stat[8] += (*it)->quadrangles.size();
    }
    for(GModel::riter it = m->firstRegion(); it != m->lastRegion(); ++it){
        stat[6] += (*it)->mesh_vertices.size();
        stat[9] += (*it)->tetrahedra.size();
        stat[10] += (*it)->hexahedra.size();
        stat[11] += (*it)->prisms.size();
        stat[12] += (*it)->pyramids.size();
    }
    stat[13] = CTX::instance()->meshTimer[0];
    stat[14] = CTX::instance()->meshTimer[1];
    stat[15] = CTX::instance()->meshTimer[2];
    if(quality){
        for(int i = 0; i < 3; i++)
            for(int j = 0; j < 100; j++)
                quality[i][j] = 0.;
        double gamma=0., gammaMin=1., gammaMax=0.;
        double eta=0., etaMin=1., etaMax=0.;
        double rho=0., rhoMin=1., rhoMax=0.;
    }
}

```

```

double disto=0., distoMin=1., distoMax=0.;
if (m->firstRegion() == m->lastRegion()){
    for(GModel::fiter it = m->firstFace(); it != m->lastFace(); ++it){
        GetQualityMeasure((*it)->quadrangles, gamma, gammaMin, gammaMax,
            eta, etaMin, etaMax, rho, rhoMin, rhoMax,
            disto, distoMin, distoMax, quality);
        GetQualityMeasure((*it)->triangles, gamma, gammaMin, gammaMax,
            eta, etaMin, etaMax, rho, rhoMin, rhoMax,
            disto, distoMin, distoMax, quality);
    }
}
else{
    for(GModel::riter it = m->firstRegion(); it != m->lastRegion(); ++it){
        GetQualityMeasure((*it)->tetrahedra, gamma, gammaMin, gammaMax,
            eta, etaMin, etaMax, rho, rhoMin, rhoMax,
            disto, distoMin, distoMax, quality);
        GetQualityMeasure((*it)->hexahedra, gamma, gammaMin, gammaMax,
            eta, etaMin, etaMax, rho, rhoMin, rhoMax,
            disto, distoMin, distoMax, quality);
        GetQualityMeasure((*it)->prisms, gamma, gammaMin, gammaMax,
            eta, etaMin, etaMax, rho, rhoMin, rhoMax,
            disto, distoMin, distoMax, quality);
        GetQualityMeasure((*it)->pyramids, gamma, gammaMin, gammaMax,
            eta, etaMin, etaMax, rho, rhoMin, rhoMax,
            disto, distoMin, distoMax, quality);
    }
}
double N = stat[9] + stat[10] + stat[11] + stat[12];
stat[17] = N ? gamma / N : 0.;
stat[18] = gammaMin;
stat[19] = gammaMax;
stat[20] = N ? eta / N : 0.;
stat[21] = etaMin;
stat[22] = etaMax;
stat[23] = N ? rho / N : 0.;
stat[24] = rhoMin;
stat[25] = rhoMax;
stat[46] = N ? disto / N : 0.;
stat[47] = distoMin;
stat[48] = distoMax;
}
#ifdef HAVE_NO_POST
stat[26] = PView::list.size();
for(unsigned int i = 0; i < PView::list.size(); i++) {
    PViewData *data = PView::list[i]->getData(true);
    stat[27] += data->getNumPoints();
    stat[28] += data->getNumLines();
    stat[29] += data->getNumTriangles();
    stat[30] += data->getNumQuadrangles();
    stat[31] += data->getNumTetrahedra();
    stat[32] += data->getNumHexahedra();
    stat[33] += data->getNumPrisms();
    stat[34] += data->getNumPyramids();
    stat[35] += data->getNumStrings2D() + data->getNumStrings3D();
}
#endif
}

```

```

static bool TooManyElements(GModel *m, int dim)
{
    if(CTX::instance()->expertMode || !m->getNumVertices()) return false;
    // try to detect obvious mistakes in characteristic lengths (one of
    // the most common cause for erroneous bug reports on the mailing
    // list)
    double sumAllLc = 0.;
    for(GModel::viter it = m->firstVertex(); it != m->lastVertex(); ++it)
        sumAllLc += (*it)->prescribedMeshSizeAtVertex() * CTX::instance()->mesh.LcFactor;
    sumAllLc /= (double)m->getNumVertices();
    if(!sumAllLc || pow(CTX::instance()->lc / sumAllLc, dim) > 1.e10)
        return !Msg::GetBinaryAnswer
            ("Your choice of characteristic lengths will likely produce a very\n"

            "large mesh. Do you really want to continue?\n\n"
            "(To disable this warning in the future, select `Enable expert mode`\n"
            "in the option dialog.)",
            "Continue", "Cancel");
    return false;
}

static bool CancelDelaunayHybrid(GModel *m)
{
    if(CTX::instance()->expertMode) return false;
    int n = 0;
    for(GModel::riter it = m->firstRegion(); it != m->lastRegion(); ++it)
        n += (*it)->getNumMeshElements();
    if(n)
        return !Msg::GetBinaryAnswer
            ("You are trying to generate a mixed structured/unstructured grid using\n"
            "the 3D Delaunay algorithm. This algorithm cannot guarantee that the\n"
            "final mesh will be conforming. You should probably use the 3D Frontal\n"
            "algorithm instead. Do you really want to continue?\n\n"
            "(To disable this warning in the future, select `Enable expert mode`\n"
            "in the option dialog.)",
            "Continue", "Cancel");
    return false;
}

static void Mesh1D(GModel *m)
{
    if(TooManyElements(m, 1)) return;
    Msg::StatusBar(1, true, "Meshing 1D...");
    double t1 = Cpu();
    std::for_each(m->firstEdge(), m->lastEdge(), meshGEdge());
    double t2 = Cpu();
    CTX::instance()->meshTimer[0] = t2 - t1;
    Msg::Info("Mesh 1D complete (%g s)", CTX::instance()->meshTimer[0]);
    Msg::StatusBar(1, false, "Mesh");
}

static void PrintMesh2dStatistics(GModel *m)
{
    FILE *statreport = 0;
    if(CTX::instance()->createAppendMeshStatReport == 1)
        statreport = fopen(CTX::instance()->meshStatReportFileName.c_str(), "w");
    else if(CTX::instance()->createAppendMeshStatReport == 2)
        statreport = fopen(CTX::instance()->meshStatReportFileName.c_str(), "a");
    else return;
    double worst = 1, best = 0, avg = 0;

```



```

}

// collapseSmallEdges(*m);
double t2 = Cpu();
CTX::instance()->meshTimer[1] = t2 - t1;
Msg::Info("Mesh 2D complete (%g s)", CTX::instance()->meshTimer[1]);
Msg::StatusBar(1, false, "Mesh");
PrintMesh2dStatistics(m);
}
static void FindConnectedRegions(std::vector<GRegion*>&delaunay,
                                std::vector<std::vector<GRegion*>>&connected)
{
    // FIXME: need to split region vector into connected components here!
    connected.push_back(delaunay);
}
static void Mesh3D(GModel *m)
{
    if(TooManyElements(m, 3)) return;
    Msg::StatusBar(1, true, "Meshing 3D...");
    double t1 = Cpu();
    // mesh the extruded volumes first
    std::for_each(m->firstRegion(), m->lastRegion(), meshGRegionExtruded());
    // then subdivide if necessary (unfortunately the subdivision is a
    // global operation, which can require changing the surface mesh!)
    SubdivideExtrudedMesh(m);
    // then mesh all the non-delaunay regions
    std::vector<GRegion*> delaunay;
    std::for_each(m->firstRegion(), m->lastRegion(), meshGRegion(delaunay));
    // warn if attempting to use Delaunay for mixed meshes
    if(delaunay.size() && CancelDelaunayHybrid(m)) return;
    // and finally mesh the delaunay regions (again, this is global; but
    // we mesh each connected part separately for performance and mesh
    // quality reasons)
    std::vector<std::vector<GRegion*>> connected;
    FindConnectedRegions(delaunay, connected);
    for(unsigned int i = 0; i < connected.size(); i++){
        MeshDelaunayVolume(connected[i]);
    }
    double t2 = Cpu();
    CTX::instance()->meshTimer[2] = t2 - t1;
    Msg::Info("Mesh 3D complete (%g s)", CTX::instance()->meshTimer[2]);
    Msg::StatusBar(1, false, "Mesh");
}
void OptimizeMeshNetgen(GModel *m)
{
    Msg::StatusBar(1, true, "Optimizing 3D with Netgen...");
    double t1 = Cpu();
    std::for_each(m->firstRegion(), m->lastRegion(), optimizeMeshGRegionNetgen());
    double t2 = Cpu();
    Msg::Info("Mesh 3D optimization with Netgen complete (%g s)", t2 - t1);
    Msg::StatusBar(1, false, "Mesh");
}
void OptimizeMesh(GModel *m)
{
    Msg::StatusBar(1, true, "Optimizing 3D...");
    double t1 = Cpu();
    std::for_each(m->firstRegion(), m->lastRegion(), optimizeMeshGRegionGmsh());
}

```

```

double t2 = Cpu();
Msg::Info("Mesh 3D optimization complete (%g s)", t2 - t1);
Msg::StatusBar(1, false, "Mesh");
}
void AdaptMesh(GModel *m)
{
    Msg::StatusBar(1, true, "Adapting 3D Mesh...");
    double t1 = Cpu();
    if(CTX::instance()->lock) {
        Msg::Info("I'm busy! Ask me that later...");
        return;
    }
    CTX::instance()->lock = 1;
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    std::for_each(m->firstRegion(), m->lastRegion(), adaptMeshGRegion());
    double t2 = Cpu();
    Msg::Info("Mesh Adaptation complete (%g s)", t2 - t1);
    Msg::StatusBar(1, false, "Mesh");
}
void GenerateMesh(GModel *m, int ask)
{
    if(CTX::instance()->lock) {
        Msg::Info("I'm busy! Ask me that later...");
        return;
    }
    CTX::instance()->lock = 1;
    Msg::ResetErrorCounter();
    int old = m->getMeshStatus(false);
    // Initialize pseudo random mesh generator with the same seed
    srand(1);
    // Change any high order elements back into first order ones
    SetOrder1(m);
    // 1D mesh
    if(ask == 1 || (ask > 1 && old < 1)) {
        std::for_each(m->firstRegion(), m->lastRegion(), deMeshGRegion());
        std::for_each(m->firstFace(), m->lastFace(), deMeshGFace());
        Mesh1D(m);
    }
    // 2D mesh
    if(ask == 2 || (ask > 2 && old < 2)) {
        std::for_each(m->firstRegion(), m->lastRegion(), deMeshGRegion());
        Mesh2D(m);
    }
    // 3D mesh
    if(ask == 3) {
        Mesh3D(m);
    }
}

```

```

// Orient the surface mesh so that it matches the geometry
if(m->getMeshStatus() >= 2)
    std::for_each(m->firstFace(), m->lastFace(), orientMeshGFace());
// Optimize quality of 3D tet mesh
if(m->getMeshStatus() == 3){
    for(int i = 0; i < std::max(CTX::instance()->mesh.optimize,
                               CTX::instance()->mesh.optimizeNetgen); i++){
        if(CTX::instance()->mesh.optimize > i) OptimizeMesh(m);
        if(CTX::instance()->mesh.optimizeNetgen > i) OptimizeMeshNetgen(m);
    }
}
// Subdivide into quads or hexas
if(m->getMeshStatus() == 2 && CTX::instance()->mesh.algoSubdivide == 1)
    RefineMesh(m, CTX::instance()->mesh.secondOrderLinear, true);
else if(m->getMeshStatus() == 3 && CTX::instance()->mesh.algoSubdivide == 2)
    RefineMesh(m, CTX::instance()->mesh.secondOrderLinear, false, true);
// Create high order elements
if(m->getMeshStatus() && CTX::instance()->mesh.order > 1)
    SetOrderN(m, CTX::instance()->mesh.order, CTX::instance()->mesh.secondOrderLinear,
              CTX::instance()->mesh.secondOrderIncomplete);
Msg::Info("%d vertices %d elements",
           m->getNumMeshVertices(), m->getNumMeshElements());
Msg::PrintErrorCounter("Mesh generation error summary");
CTX::instance()->lock = 0;
CTX::instance()->mesh.changed = ENT_ALL;
}

```