УДК 004.42

Програмний комплекс автоматизації скінченно-елементної дискретизації двовимірних областей

Ярослав Соколовський[1], Ольга Сикала[2]

[1] д. т. н., професор, Національний лісотехнічний університет України, вул. Генерала Чупринки, 103, Львів, e-mail: sokolowskyy@ukr.net
[2] Національний лісотехнічний університет України, вул. Генерала Чупринки, 103, Львів, e-mail: helga8p@gmail.com

Розроблено програмний комплекс для автоматизації скінченно-елементної дискретизації двовимірних областей мовою програмування C# у середовищі Microsoft Visual Studio (.NET Framework). У рамках об'єктно-орієнтованого підходу наведено інформаційну модель системи у вигляді спроектованих графічних діаграм UML варіантів використання, класів і відношень між ними. Створений графічний інтерфейс користувача дозволяє задавати параметри тріангуляції двовимірних областей і контролювати зміни геометричних розмірів елементів розбиття з метою згущення сітки у заданих місцях області. Розроблені класи відображають сутність об'єктно-орієнтованої реалізації ітераційних методів тріангуляції на основі критерію Делоне. Це створює можливість інтегрування розробленого програмного забезпечення до існуючих CAE/CAD/CAM систем із метою розширення їх функціональних можливостей.

Ключові слова: тріангуляція, критерій Делоне, ітераційний метод, метод скінченного елемента, об'єктно-орієнтоване програмування.

Вступ. Розроблення сучасних інформаційних технологій скінченно-елементного аналізу передбачає автоматизацію дискретизації розрахункової області на скінченне число 2D або 3D елементів. Отримана дискретна область характеризується такими об'єктами, як вузли, ребра, грані та скінченні елементи, а їх розподіл і характерні геометричні властивості визначаються адитивним чином з урахуванням апостеріорної чи апріорної інформації щодо вихідних даних крайових задач або особливостей їх розв'язання. З одного боку, автоматизація дискретизації області суттєво впливає на обчислювальну складність методу скінченних елементів (МСЕ), а з іншого, її вирішення зумовлює використання іншого комплексу алгоритмів і методів, ніж ті, що використовуються безпосередньо в чисельній реалізації МСЕ. Тому актуальною задачею є створення об'єктно-орієнтованого програмного комплексу для автоматизації дискретизації двовимірних областей. Програмна реалізація автоматизації конструювання сіток, насамперед, має забезпечувати максимально точну апроксимацію границь області та контролювати розміри елементів сітки з метою її згущення в підобластях різкої зміни цільових функцій або їх похідних. Такі програмні пакети повинні мати зручний інтерфейс користувача з широкими можливостями графічної візуалізації скінченно-елементного розбиття, ефективних програм зберігання й імпортування даних.

1. Аналіз існуючих досліджень

Попередній аналіз різних програмних засобів дискретизації плоских і просторових областей свідчить про суттєвий вплив вибраних алгоритмів розбиття на обумовленість алгебраїчних систем МСЕ і на його обчислювальну складність у цілому [1-3]. В основному використовуються методи дискретизації на основі критерію Делоне [2], але теоретичне обґрунтування побудови дискретних моделей реальних тіл є далеке до завершення. Програмні комплекси поділяються на комерційні програми та безкоштовні програми з відкритим кодом для академічного використання [4, 5]. У деяких програмних комплексах не створені візуальні інтерфейси [2, 5, 6], а також відсутні засоби щодо безпосереднього корегування розмірів елементів розбиття. В окремих програмах використовуються нестандартні форми обміну даними. Окрім того, переважна більшість таких програмних продуктів використовує програмні коди, написані на C++ або FORTRAN 77, що обмежує інваріантність програм, використання готових компонентів, можливу взаємодію. У роботах [2, 7, 8] перераховуються проблеми тривимірної дискретизації, пов'язані з труднощами адаптації та перенесенням у тримірний простір алгоритмів розбиття на площині, адже відомо про те, що неможливо заповнити простір правильними тетраедрами та розбити правильний багатогранник на тетраедри без використання додаткових вузлів. У той же час площину не завжди можна заповнити правильними трикутниками, але будь-який багатогранник на площині можна розбити на трикутники без використання додаткових вузлів. Тому задача дискретизації областей, зокрема тріангуляції, формулюється як розбиття заданої області на елементи, що характеризуються найкращими апроксимаційними властивостями. У низці робіт запропоновані різні способи оцінки дискретизації, як за надійністю, так і за об'ємом обчислювальних затрат [9, 10].

Розміщення вузлів в області дискретизації може здійснюватися різними способами. Зокрема, у роботі [11] використовувалися генератори псевдовипадкових чисел із заданими умовами розподілу. Інший клас методів генерування вузлів базується на мінімізації функціоналів і розв'язанні диференціальних рівнянь [12]. Використовуються також ітераційні методи, однак їх застосування вимагає трудомістких процедур. Застосування алгоритмів адаптивної перебудови дискретизації дозволяє зменшити похибку МСЕ [13]. Розрізняють процедуру поділу скінченних елементів на елементи менших розмірів (h-версія), зміну координат вузлів області (r-версія) та їх комбінації (hr-версія). Питання вибору критеріїв адаптації області дискретизації для застосування таких процедур розглядаються у роботі [14]. Апроксимаційні властивості елементів дискретизації, зокрема трикутників і тетраедрів, наведені у праці [2].

Найуніверсальнішими підходами дискретизації двовимірних областей є тріангуляція області. Методи тріангуляції та характеристики трикутників розбиття можуть суттєво впливати на точність наступних апроксимацій, обумовленість алгебраїчних систем та обчислювальну складність МСЕ в цілому. За принципом побудови методи тріангуляції можна поділити на дві групи — прямі та ітераційні [2]. Головною перевагою прямих методів є швидкість і надійність розбиття, оскільки

тріангуляція будується в один етап, а координати усіх вузлів і зв'язки між ними є відомі. Ці методи можна умовно поділити на дві взаємозв'язані групи: методи на основі шаблонів (розбиття області заданого виду) та методи відображення (дозволяють у вигляді наявності взаємооднозначного відображення перейти від областей канонічної геометричної форми до областей довільного вигляду). Через значні обмеження прямих методів використання алгоритмів і програмних засобів їх реалізації розроблено небагато. Ітераційні методи тріангуляції отримали найбільший розвиток. Для їх реалізації розроблені декілька різних підходів, які можна поділити на три групи: методи на основі критерію Делоне, граничної корекції та вичерпування. Для двовимірних областей методи Делоне дозволяють швидко й ефективно здійснювати тріангуляцію з високою якістю. Огляд алгоритмів побудови тріангуляції Делоне, їх класифікація, опис трудомісткості й особливостей реалізації наведені у роботах[15, 16]. Розглядаються різні структури даних подання тріангуляції.

2. Алгоритмічні аспекти ітераційних алгоритмів тріангуляції у МСЕ

У програмному забезпеченні автоматизації побудови тріангуляції використано ітеративний алгоритм «видаляй та будуй» [15]. Усі ітераційні алгоритми базуються на ідеї послідовного додавання вузлів у частково побудовану тріангуляцію Делоне. Їх складність полягає: у пошуку трикутника, в який ітераційно додається вузол; побудові нових трикутників; відповідній перебудові структури тріангуляції, зумовленій перевіркою умови Делоне. Якщо доданий вузол не потрапляє всередину області тріангуляції, то для спрощення алгоритму побудови нових трикутників використовується суперструктура [17]. Її суть полягає у внесенні додаткових вузлів в область тріангуляції з метою покриття усіх попередніх вузлів «новою» тріангуляцією. Теоретично, трудомісткість перебудов тріангуляції пов'язана з локальною перевіркою умов Делоне та складає $O(N)$. Однак, згідно роботи [15], для реальних даних потрібно здійснити не більше трьох перебудов. Тому усі ітеративні алгоритми відрізняються один від одного лише процедурою пошуку максимального трикутника. Вибір алгоритму «будуй та видаляй» зумовлено тим, що у ньому не потрібна процедура багатократної перебудови області тріангуляції. За допомогою цього алгоритму відразу будуються усі необхідні елементи розбиття, оскільки у разі додавання нового вузла вилучаються усі трикутники, у яких всередину описаних кіл потрапляє новий вузол. Видалені трикутники неявно утворюють деякий полігон, у якому будується нова тріангуляція шляхом з'єднання нового вузла з вузлами полігону.

Нехай двовимірна область D розбивається на K трикутних елементів, що не перетинаються $D_k^h, k=\overline{1,K}$. Їх об'єднання $D^h=\bigcup_{k=1}^{K} D_k^h$ утворює область дискретизації вихідної області D. Можливі два випадки, зокрема $D^h=D$. Але допускається $D=D^h+D_1^h\approx D^h$. Тут D_1^h характеризує похідну апроксимації області D та є величиною $O\left(h^{-d\gamma}\right)$, де d — розмірність D, γ — порядок

апроксимації області, $h = O\left(k^{-d}\right)$ — характерний крок розбиття, зокрема для $K \to \infty$, $h \to 0$.

Під побудовою тріангуляції розуміється задача з'єднання заданих в області D^h множини вузлів $\left\{x_i \middle| i \in J_k\right\}$ відрізками таким чином, щоб кожний вузол став вершиною трикутника з D^h, тобто:

$$D \approx D^h = \bigcup_{k=1}^{K} D_k^h, \quad D_k^h \bigcap D_e^h = \begin{cases} \varnothing, \\ x, \forall D_k^h, D_k^h \in D^h, k \neq e, \\ \left\{x_k, x_e\right\}. \end{cases} \tag{1}$$

Тут x і $\left\{x_k, x_e\right\}$ — спільний вузол і спільна сторона трикутників D_k^h та D_e^h.

Важливою процедурою у процесі побудови тріангуляції є перевірка критерію Делоне. Для забезпечення виконання цього критерію застосовано спосіб перевірки сум прилеглих кутів трикутників [15].

Згідно основної концепції МСЕ у формулюванні Рітца або Гальоркіна [1, 3] наближений розв'язок u^h в області дискретизації D^h шукається у вигляді лінійної комбінації фінітних функцій φ_i, $i = \overline{1, J}$, які складають базис скінченновимірного простору R_h.

$$u^h = \sum_{i=1}^{J} v_i^k \varphi_i, \tag{2}$$

де $v^h = \left\{v_i^k\right\}$ — шуканий вектор порядку J, що визначається як розв'язок алгебраїчної системи рівнянь МСЕ

$$A^h v^h = f^h. \tag{3}$$

Квадратна матриця A^h і відомий вектор f^h такого ж порядку визначаються формулами:

$$A^h = \sum_{k=1}^{K} A_k^h, \quad f^h = \sum_{k=1}^{K} f_k^h. \tag{4}$$

Ненульові елементи A_k^h та f_k^h мають вигляд:

$$a_{ij}^{(K)} = a\left(\sum_{j=1}^{J} \varphi_j, \varphi_i\right)^{(k)}, \qquad f_i^{(k)} = \left(f, \varphi_i\right)^{(k)}. \tag{5}$$

Тут верхній індекс K у білінійній або лінійній формах означає номер відповідного скінченного елемента D_k^h, за яким ведеться інтегрування для визначення величини $a_{ij}^{(k)}$ та $f_i^{(k)}$, тобто

$$a(\cdot,\cdot)^{(k)} = \int\limits_{D_K^h} = \sum_k a(\cdot,\cdot)^{(k)}\ , \qquad a(\cdot,\cdot)^{(k)} = \int\limits_{D_k^h}\ .$$

Нехай M_k число базисних функцій, відмінних від нуля на елементі D_k^h, а J_k — кількість відповідних номерів функцій φ_i. Тоді апроксимацію на кількісному елементі згідно (2) запишемо у вигляді:

$$U^h{}_{|D_k^h} = \sum_{i\in J_k} v_i^h \varphi_i = \sum_{j=1}^{M_k} v_j^h \varphi_j\ , \tag{6}$$

де *j* відповідають локальним номерам відмінних від нуля функції φ_i на елементі D_k^h.

Позначимо через $\tilde{\mathbf{A}}_k$ та $\tilde{\mathbf{f}}_k^h$ квадратну матрицю та вектор розмірності M_k, які складаються з ненульових елементів відповідно A_k і f_k^h та визначені за формулами (5). Тоді між описаними матрицями та векторами різних розмірностей існують такі взаємозв'язки:

$$\mathbf{A}_k^h = \mathbf{P}_k \tilde{\mathbf{A}}_k^h \mathbf{P}_k^{\mathrm{T}} \quad \text{та} \quad \mathbf{f}_k^h = \mathbf{P}_k \tilde{\mathbf{f}}_k^{\mathrm{T}}\ . \tag{7}$$

Тут $\mathbf{P}_k$ — прямокутна матриця розмірності $\left(J \times M_k\right)$, «Т» — операція транспонування.

Для конкретизації поставленої задачі дискретизації розглянемо лагранжеві скінченні елементи. Тоді v_i^h відповідатимуть наближеним значенням шуканого розв'язку (2) у точках області $x_i \in D^k$, а базисна функція визначається $\varphi_i\left(x_j\right) = \delta_i^j$ (δ_i^j — символ Кронекера). Нехай x_i як вузли дискретизації будуть вершинами тріангуляції трикутників в D^h. Тоді для кожного вузла x_i можна записати рівняння МСЕ у вигляді (2).

3. Проектування програмного забезпечення. Модель варіантів використання

Розроблений програмний засіб для реалізації ітераційного алгоритму тріангуляції двовимірної області має низку функцій, які ґрунтуються на принципах об'єктно-орієнтованого програмування [18, 19]. Перед користувачем відкривається можливість здійснювати тріангуляцію заданого згущення. Є доступний подальший контроль і локальна зміна геометричних розмірів сітки. Це можна виконати як в ручному режимі (користувач сам обирає місце, куди потрібно додати вузли), так і автоматизовано (обравши одну з двох можливих структур дискретизації). Перераховані функції програмного засобу відображені на діаграмі варіантів використання , яку подано на рис. 1.

Після завершення тріангуляції користувач може переглянути масиви необхідних параметрів кожного побудованого елемента, зокрема номери трикутників, вузлів і

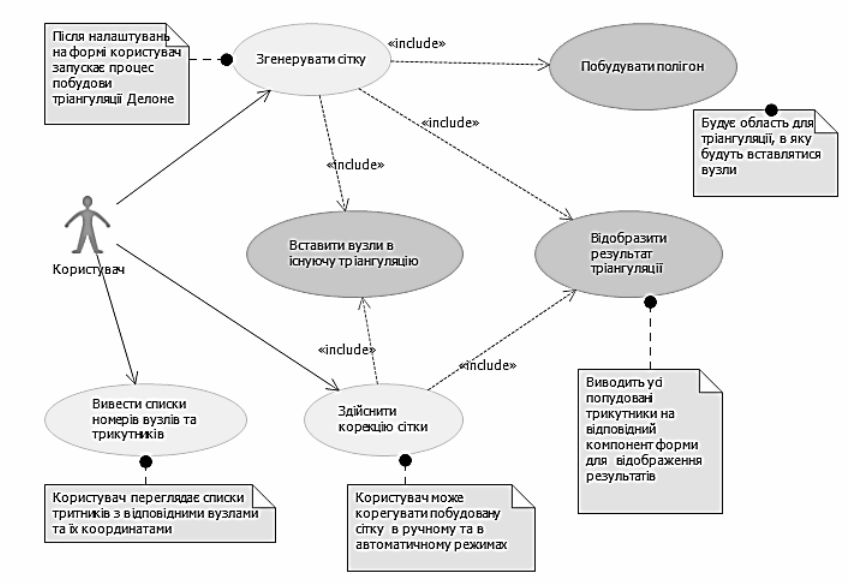

Рис. 1. Діаграма варіантів використання програмної системи

їх координати. На будь-якому етапі тріангуляції є можливість скористатися перевіркою умови Делоне. Це дозволяє переконатися в коректності побудованої сітки.

4. Проектування класів програмного забезпечення

Для побудови сітки було обрано структура даних «трикутники–вузли» [15]. Кожен трикутник характеризується трьома вузлами, а вузол, своєю чергою, двома координатами (x, y). Також у цій структурі фігурує поняття ребра, як посилання на дві вершини. Для відображення такої структури були розроблені класи, опис і відношення між ними (рис. 2).

Варто розпочати з опису вузлів. Для цього розроблено клас Point. Кожен вузол характеризується координатами (x, y). Також у клас вузлів внесено поле GlobalIndex, яке зберігатиме «глобальний» індекс нумерації вузлів. Цей індекс використовуватиметься для перебору усіх вузлів області тріангуляції, оскільки тоді доведеться порівнювати вузли лише за індексом, а не координатами. Що стосується перевірки на рівність двох вузлів, то з метою уникнення потрапляння в область тріангуляції під іншим індексом вузла з ідентичними координатами передбачено метод Equals(Point p), який порівнює два вузли на основі використання стандартного інтерфейсу IEquatable <> платформи .NET Framework. В реалізації розробленого класу такий метод подано у вигляді:

```
public bool Equals(Point p){ return ((X == p.X) & (Y == p.Y));}.
```

До класу Point належить перевантажений метод ToString(). Він використовується для відображення списку вузлів на формі користувача:

```
public override string ToString() { return "(" + X + "; " + Y + ")";}.
```

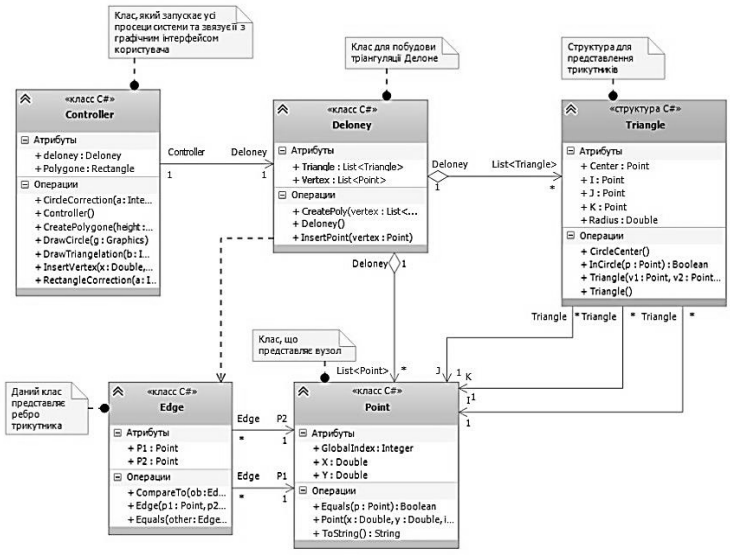

Рис. 2. Діаграма класів програмної системи

Конструктор класу Point є параметризований і формує об’єкт вузлів на основі двох координат і «глобального» індексу.

Іншим розробленим елементом програмного забезпечення є структура для подання трикутника Triangle. Об’єкт структури складається з трьох вузлів, тобто екземплярів класу Point. Ще двома важливими полями класу є Center і Radius. Вони представляють відповідно центр і радіус описаного навколо трикутника кола. Така інформація необхідна для визначення тих трикутників, які підлягають перебудові, зумовленій внесенням нового вузла в область тріангуляції. Якщо поля вузлів ініціалізуються зі створенням об’єкта трикутника, то значення полів Center і Radius описаного кола визначаються методом CircleCenter() на основі координат вузлів трикутника.

```
public void CircleCenter(){
double m1, m2, x1, x2, y1, y2;
if (System.Math.Abs(j.Y - i.Y) < double.Epsilon){
m2 = -(k.X - j.X) / (k.Y - j.Y);
x2 = (j.X + k.X) * 0.5;
y2 = (j.Y + k.Y) * 0.5;
//Визначення центру кола (xc,yc) на основі вузлів i та j
Center.X = (j.X + i.X) * 0.5;
Center.Y = m2 * (Center.X - x2) + y2;}
else if (System.Math.Abs(k.Y - j.Y) < double.Epsilon){ …//аналогічні дії для вузлів k та j}
```

Наступним розробленим методом є InCircle(Point p). У цей метод передається вузол і визначається, чи потрапляє він в описане навколо поточного об’єкта

трикутника коло. Якщо вузол потрапляє в описане коло, то такий трикутник підлягає перебудові. Сигнатура методу є така:

```
public bool InCircle(Point p){
if (System.Math.Abs(i.Y - j.Y) < double.Epsilon && System.Math.Abs(j.Y - k.Y) < double.Epsilon)
{/*Якщо точки співпадають*/ return false;}
double dx = p.X - Center.X;
double dy = p.Y - Center.Y;
double drsqr = dx * dx + dy * dy;
return (drsqr <= (Radius*Radius));}
```

У процесі перебудови важливу роль відіграє поняття «ребра». Для покращення швидкодії програми вирішено не вносити ребро як частину трикутника, а обмежитися лише вузлами. Однак вставлення нового вузла в область тріангуляції, що вже існує, потребує видаляння трикутників зсередини області, яку слід видозмінити, а певні їх ребра залишати. У зв'язку з цією необхідністю було створено клас Edge. Основне його призначення — згрупувати вузли трикутника, які представляють одне конкретне ребро. Клас Edge отримав посилання на два вузли трикутника P1 та P2. Він зв'язаний відношенням асоціації з класом Point. Ці посилання передаються як параметри в конструктор класу. У програмі реалізовано порівняння двох ребер за допомогою стандартного інтерфейсу IComparable<Edge>[2].

За процес побудови сітки відповідає розроблений клас Deloney. Його атрибутами є список вузлів, де містяться посилання на усі вузли сітки List<Point> Vertex, і список трикутників List<Triangle> Triangles із посиланнями на побудовані елементи дискретизації — трикутники. Для проведення тріангуляції є необхідний полігон, на якому слід будувати сітку. Такий полігон будується безпосередньо в програмі за допомогою розробленого методу CreatePoly(). Результатом роботи методу буде початковий список вузлів, які розташовані на границі області тріангуляції. Усі наступні вузли проходять обов'язкову перевірку на потрапляння в заданий полігон. Ця перевірка відбувається під час вставляння нового вузла та реалізована за допомогою методу InsertPoint(Point vertex). Метод починає свою роботу зі створення тимчасового списку ребер. Туди будуть додані ребра усіх трикутників, які підлягають видаленню. Перебирається список побудованих трикутників і якщо вузол, переданий як параметр цьому методу, потрапляє в описане навколо трикутника коло, то його ребра записуються у тимчасовий список, а сам трикутник видаляється з тріангуляції. Після цього, внаслідок порівнянь, видаляються ребра які дублюються, а на основі тих, що залишилися, будуються нові трикутники з урахуванням нового вузла. Програмний код методу подано нижче.

```
public void InsertPoint(Point vertex){
List<Edge> Edges = new List<Edge>();
Vertex.Add(vertex);
for (int j = 0; j < Triangle.Count; j++){
```

```
if (Triangle[j].InCircle(vertex)){
Edges.Add(new Edge(Triangle[j].I, Triangle[j].J));
Edges.Add(new Edge(Triangle[j].J, Triangle[j].K));
Edges.Add(new Edge(Triangle[j].K, Triangle[j].I));
Triangle.RemoveAt(j);j--;}}
for (int j = Edges.Count - 2; j >= 0; j--){
for (int k = Edges.Count - 1; k >= j + 1; k--){
if (Edges[j].Equals(Edges[k])){
Edges.RemoveAt(k);
Edges.RemoveAt(j);
k--;continue;}}}
for (int j = 0; j < Edges.Count; j++)
Triangle.Add(new Triangle(Edges[j].P1, Edges[j].P2, vertex));
Edges.Clear();}
```

Хотілося б звернути увагу на вибір структури для зберігання списку вузлів, трикутників і ребер. Для збільшення швидкодії прийнято рішення відмовитися від стандартних масивів, а усі створені об'єкти розміщувати в узагальнених колекціях типу List<T> з простору імен System.Collection.Generics [18].

5. Графічний інтерфейс та його функціональність

Для забезпечення зв'язку програмної системи з графічним інтерфейсом користувача розроблено ще один клас — Controller. У процесі запуску програми користувач може вибрати параметри генерування сітки та подальшого її згущення. Всі ці дані збираються та передаються об'єктові класу Controller. Важливим методом класу є CreatePolygone(int height, int width, int density). Полігон будується виходячи з геометричних розмірів вікна для відображення процесу тріангуляції. Ці розміри передаються двом першим параметрам height і width. Третім параметром є густина сітки density, яку користувач задає самостійно. Після побудови полігону генеруються вузли, якими він заповняється та здійснюється тріангуляція шляхом їх додавання по одному. Також тут реалізовано підрахунок часу побудови тріангуляції на створеному полігоні. Метод повертає рядок із поданням часу, який буде відображатися на формі. Сигнатура розробленого методу має вигляд:

```
public String CreatePolygone(int height, int width, int density)
{ String result = "";
List<Point> PolyVertex = new List<Point>();
for (int i = 10; i < width; i += density)
for (int j = 10; j < height; j += density){
Point temp = new Point(i, j, Vertexes.Count);
AddVertex(temp);}
Polygone = new Rectangle((int)Vertexes[0].X, (int)Vertexes[0].Y,
(int)Vertexes[Vertexes.Count - 1].X - 9, (int)Vertexes[Vertexes.Count - 1].Y - 9);
PolygonExist = true;
var sw = Stopwatch.StartNew();
```

```
deloney.CreatePoly(Vertexes);
sw.Stop();
result = "Час триангуляції: " + sw.Elapsed.ToString();
return result;}
```

Наступною функціональною можливістю графічного інтерфейсу користувача є згущення побудованої тріангуляції. Тут розроблені два варіанти,зокрема перший варіант передбачає проведення згущення сітки в ручному режимі. Користувач за допомогою мишки визначає вузол в області тріангуляції. Програма зчитує координати курсору мишки та на основі них створює новий вузол і вносить до існуючої тріангуляції за допомогою методу InsertVertex(double x, double y). Згаданий метод здійснює декілька перевірок перед додаванням вершини. Спершу перевіряється, чи було створено початковий полігон для тріангуляції, потім, чи потрапляє вершина в область тріангуляції (полігон). Після цього викликається метод AddVertex(tempVertex), який перевіряє, чи не існує випадково в тріангуляції вузол із такими ж координатами. Тільки після успішної перевірки усіх перерахованих умов викликається метод класу Deloney InsertPoint(Point p).

Слід звернути увагу на кількісні характеристики сітки, які відображаються у вікні програми, зокрема кількість вузлів і трикутників побудованої тріангуляції. Також у вікні подані ще два параметри — це час здійснення тріангуляції та час її перебудови. Перший параметр визначає час, який було затрачено на отримання рівномірної сітки, а други - час, витрачений на згущення тріангуляції під час внесення до неї нового вузла. Це досить важливі показники для оцінки ефективності роботи алгоритму, оскільки чим менший є час перебудови, тим ефективніше працює алгоритм.

Другий варіант дозволяє здійснити згущення сітки в автоматичному режимі (рис. 3.).

Користувач обирає шаблон відображення полігону (коло або прямокутник) для згущення та задає відповідні параметри. У разі клікання на область тріангуляції у вікні програми автоматично додається набір вузлів (по одному за один раз).

Програмну реалізацію процесу згущення сітки як в ручному, так і в автоматизованому режимі, здійснено одним методом, програмний код якого наведено нижче.

```
public String InsertVertex(double x, double y){
var sw = Stopwatch.StartNew();
if (PolygonExist){
if (Polygone.Contains((int)x, (int)y)){Point tempVertex = new Point(x, y,
Vertexes.Count);
AddVertex(tempVertex);
deloney.InsertPoint(Vertexes[Vertexes.Count - 1]);}}
sw.Stop();
String result = "Час перебудови: " + sw.Elapsed.ToString();
return result; }
```

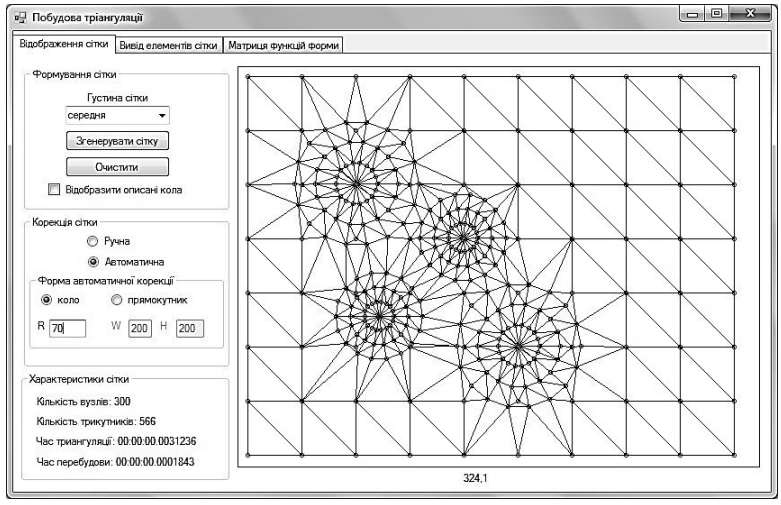

Рис. 3. Згущення тріангуляції в автоматизованому режимі

Що стосується дій, які відбуваються у вікні програми, то вони прив'язані до руху мишкою по області відображення тріангуляції. Коли користувач рухає мишкою, програма відслідковує координати розташування курсору у вікні подання результату тріангуляції та одразу відображає їх. За натискання на кнопку мишки обробляється відповідна подія. В обробнику події здійснюється перевірка на обраний режим корекції сітки. Якщо поточним є ручне згущення, то викликається метод InsertVertex() класу Controller. Як параметри в цей метод передаються поточні координати розташування курсору, створюється вузол із відповідними координатами та додається до сітки тріангуляції. У випадку обрання автоматизованого згущення сітки викликається метод CircleCorrection() або Rectangle Correction(). Який із двох методів викликатиметься, знову ж таки, залежить від вибору користувача.

Для виклику методу CircleCorrection() необхідно вказати радіус описаного кола. Центром такого кола служитиме точка, в якій знаходиться курсор, коли користувач натиснув на кнопку мишки. У метод RectangleCorrection() передається значення довжини та ширини прямокутника, в якому буде проводитися згущення. Програма, отримавши підобласть у вигляді кола чи прямокутника, в яких слід проводити згущення, послідовно додає точки в тріангуляцію. Фрагмент програмного коду таких дій наведено нижче.

```
private void pictureBoxMouseDown(object sender, MouseEventArgs e)
{ if (released)released = false;
if (monual.Checked)label4.Text = manager.InsertVertex(e.X, e.Y);
else if (automatic.Checked){
if(circle.Checked) manager.CircleCorrection(e.X, e.Y,
Convert.ToInt32(txtR.Text));
```

```
else if(rectan.Checked) manager.RectangleCorrection(e.X, e.Y,
Convert.ToInt32(txtW.Text),
Convert.ToInt32(txtH.Text));}
Refresh();}
```

Після виконання усіх описаних вище дій, програма відображає результат процесу тріангуляції у вікні . Ці дії відбуваються у разі виникнення події Paint(). Щоразу як відбуваються якісь зміни у вікні відображення тріангуляції, метод-обробник події запускає метод DrawTriangelation() класу Controller. Основним параметром цього методу є графічний об'єкт Graphics g, який буде будувати кластер трикутників. Уся інша необхідна інформація міститься безпосередньо в об'єкті класу Controller. Для побудови тріангуляції необхідно мати як мінімум три вузли, тому проводиться перевірка на їх наявність. Далі перебираються усі створені трикутники та відбувається прорисовування.

```
public void DrawTriangelation(Image b, Graphics g, Pen myPen)
{ if (deloney.Vertex.Count > 2)
{foreach (Triangle currentTriangle in deloney.Triangle)
{ g.DrawLine(myPen, (float)currentTriangle.I.X, (float)currentTriangle.I.Y,
(float)currentTriangle.J.X, (float)currentTriangle.J.Y);
g.DrawLine(myPen, (float)currentTriangle.J.X, (float)currentTriangle.J.Y,
(float)currentTriangle.K.X, (float)currentTriangle.K.Y);
g.DrawLine(myPen, (float)currentTriangle.I.X, (float)currentTriangle.I.Y,
(float)currentTriangle.K.X, (float)currentTriangle.K.Y); } }
for (int i = 0; i < deloney.Vertex.Count; i++)
g.DrawEllipse(Pens.Firebrick, (float)deloney.Vertex[i].X - 2,
(float)deloney.Vertex[i].Y - 2, 4, 4); }
```

Окрім графічного подання результатів тріангуляції користувач має змогу переглянути параметри усіх вузлів, включених у тріангуляцію, та елементів (трикутників). Для цього слід перейти на вкладу «Параметри елементів сітки». Кожен вузол має свій «глобальний» індекс щодо усієї тріангуляції, а трикутник характеризується трьома вузлами. У табл. 1 наведено характеристики деякої частини побудованої тріангуляції.

Таблиця 1

Характеристики елементів тріангуляції

№ елемента	Номер і координати вершини		
	1	2	3
1	Вершина 2 (10; 190)	Вершина 1 (10; 100)	Вершина 7 (100; 190)
2	Вершина 1 (10; 100)	Вершина 6 (100; 100)	Вершина 7 (100; 190)
3	Вершина 3 (10; 280)	Вершина 2 (10; 190)	Вершина 8 (100; 280)
4	Вершина 2 (10; 190)	Вершина 7 (100; 190)	Вершина 8 (100; 280)
5	Вершина 4 (10; 370)	Вершина 3 (10; 280)	Вершина 9 (100; 370)
6	Вершина 3 (10; 280)	Вершина 8 (100; 280)	Вершина 9 (100; 370)
7	Вершина 7 (100; 190)	Вершина 6 (100; 100)	Вершина 12(190; 190)

Висновки. Розроблено прикладне програмне забезпечення для автоматизації процесу тріангуляції двовимірних областей. Створений графічний інтерфейс дозволяє здійснювати згущення сітки дискретизації області у ручному й автоматизованому режимах, а також отримувати кількісні характеристики тріангуляції для оцінки ефективності роботи алгоритмів.

Побудовані класи, відношення між ними та їх методи відображають сутність об'єктно-орієнтованої реалізації ітераційних алгоритмів тріангуляції двовимірних областей на основі використання критерію Делоне. Це створює можливість повторного використання класів та інтегрування розробленої програми до існуючих програмних систем автоматизованого скінченно-елементного розрахунку.

Література

[1] *Ильин В. П.* Методы и технологии конечных элементов. — Новосибирск, 2007. — 370 с.
[2] *Щеглов И. А.* Дискретизация сложных областей и трехмерных областей для решения задач математического моделирования. Автореф. дисс….к-та физ.-мат. наук: 05.13.15-мат. моделирование, численные методы и комплексы программ / Московский государственный технический университет имени Н. Э. Баумана — Москва, 2010. — 18 с.
[3] *Савула Я. Г.* Числовий аналіз задач математичної фізики варіаційними методами. — Львів: Вид-во ЛНУ ім. Франка, 2004. — 222 с.
[4] http://members.chav.ca/bjoe/.
[5] http://www.geuz.org/gmsh/.
[6] http://www.hpfem.jku.ua/netgen.
[7] *Joe B.* Construction of three-dimensional improved-quality triangulations using local transformation // SIAM Journal on Scientific Computing. — 1995. — Vol.16. — P. 1292-1307.
[8] *Montenegro R.* Quality improvement of surface triangulations // 14th International Meshing Roundtable: proceedings. Sandia National Laboratories. — 2005. — P. 469-484.
[9] *Branets L., Carey G. F.* Cell quality meric and variational grid smoothing algorithm // 12th International Meshin Roundtable: proceeding. Sandia Nationaal Laboratories. — 2003. — P. 371-390.
[10] *Parthasarathy V. N., Graichen C. M., Hathaway A. F.* A comparison of tetrahedron quality measures // Finite Elements in Analysis and Design. — 1993. — № 15. — P. 255-261.
[11] *Дворников М. В., Тишкин В. Ф., Филатов А. Ю.* Триангуляция произвольной многосвязной области со сложной границей.// , Препринт РАН, Ин-т моделирования. Москва, С. 23. (1995)
[12] *Яненко Н. Н., Данаев Н. Т., Лисейкин В. Д.* О вариационном методе построения сеток // Численные методы механики сплошной среды. — Новосибирск, 1987. — Т. 8, № 4. — С. 157-163.
[13] *Копысов С. П., Новиков А. К.* Сравнение r- и h-версий МКЭ на примере задач теории упругости // Актуальные проблемы прикл. математики и механики. 3-7 февраля 2003 г. — Екатеринбург, 2003. — С. 44-45.
[14] Неструктурированные адаптивные сетки для задач математической физики (обзор) / *Л. В. Круглякова, А. В. Неледова, В. Ф. Тишкин, А. Ю. Филатов* // Матем. моделирование. — 1998. — Т. 10, № 3. — С. 93-116.
[15] *Скворцов А. В.* Обзор алгоритмов триангуляции Делоне // Вычислительные методы и программирование. — 2002. — Т. 3. — С. 14-39.
[16] *Скворцов А. В.* Триангуляция Делоне и её применение. — Томск: Изд-во Том. ун-та, 2002. — С. 128.
[17] *Скворцов А. В., Костюк Ю. Л.* Эффективнее алгоритмы построения триангуляции Делоне // Геоинформатика. Теория и практика. Вып. 1. — Томск: Изд-во Томского ун-та, 1998. — С. 22-47.
[18] *Троелсен Э.* Язык программирования C# 2010 и платформа .NET 4.0: 5-е узд.; пер. с англ. — Москва: ООО «И. Д. Вильямс», 2011. — С. 1392.
[19] Объектно-ориентированный анализ и проектирование с примерами приложений / *Г. Буч, Р. Максимук, М. Энгл, Б. Янг.* — Киев: Изд-во «Вильямс», 2008. — 720 с.

ISSN 1816-1545 Фізико-математичне моделювання та інформаційні технології
2014, вип. 19, 176-189

Software complex for automation of finite-element discretization of 2-D domains

Yaroslav Sokolovskyy, Olha Sykala

Software complex for automation of finite-element discretization of two-dimensional domains is developed using C# programming language in Microsoft Visual Studio (.NET Framework). Within the object-oriented approach, an information model of the system in the form of designed graphic UML diagrams of cases of the use, classes and relations between them is presented. The created graphical user interface allows you to set the parameters of two-dimensional triangulation areas and to monitor the changes in the geometrical dimensions of the elements with the aim of thickening the grid in the given places of the area. The developed classes reflect the gist of object-oriented implementation of iterative methods for triangulation based on the Delaunay criterion. This creates the possibility of integrating the developed software to existing CAE/CAD/CAM systems in order to extend their functionality.

Программный комплекс автоматизации конечно-элементной дискретизации двумерных областей

Ярослав Соколовский, Ольга Сыкала

Разработан программный комплекс для автоматизации конечно-элементной дискретизации двумерных областей на языке программирования C# в среде Microsoft Visual Studio (.NET Framework). В рамках объектно-ориентированного подхода приведена информационная модель системы в виде спроектированных графических диаграмм UML вариантов использования, классов и отношений между ними. Созданый графический интерфейс пользователя позволяет задавать параметры триангуляции двумерных областей и контролировать изменения геометрических размеров элементов разбиения с целью сгущения сетки в заданных местах области. Разработанные классы отражают сущность объектно-ориентированной реализации итерационных методов триангуляции на основе критерия Делоне. Это создает возможность интегрирования разработанного программного обеспечения с существующими CAE/CAD/CAM систем с целью расширения их функциональных возможностей.

Представлено доктором технічних наук Б. Гайвась

Отримано 02.06.14