Междунар. конф. "Интеллектуальное управление: новые интеллектуальные технологии в задачах управления (ICIT'99)". – 1999. – С.4. 11. Грицюк Ю.І. Оптимізація технологічного плану розкрою плитних деревних матеріалів на меблеві заготовки. – Львів: Видавничий дім «Панорама», 2004. – 484 с. 12. Грицюк Ю.І. Проблема моделювання карт і оптимізації плану розкрою плитних деревних матеріалів на меблеві заготовки // Зб. наук.-техн. пр. Науковий Вісник НЛТУ України. – 2006. – Вип. 16.7. – С. 102–110. 13. Джон Х. Холланд. Генетические алгоритмы // В мире науки. – 1992. – №9 – №10. – С. 32–40. 14. Лебедев Б.К. Методы поисковой адаптации в задачах автоматизированного проектирования СБИС: монография. – Таганрог: Изд-во ТРТУ, 2000. – 192 с. 15. Лебедев В. Б. Планирование СБИС методом адаптивного поиска // Перспективные информационные технологии и интеллектуальные системы. – 2000. – №4. – С. 55–65. 16. Лебедев В.Б. Планирование СБИС методом генетического поиска // Перспективные информационные технологии и интеллектуальные системы. – 2000. – №2 – С. 97–105. 17. Мукачева Э.А. Модели и методы расчета раскроя – упаковки геометрических объектов // Э.А. Мукачева, М.А. Верхотуров, В.В. Мартынов. – Уфа: УГАТУ. – 1999. – 217 с. 18. Мукачева А.С. Генетический алгоритм поиска минимума в задачах двумерного гильотинного раскроя // А.С. Мукачева, А. В. Чиглинцев // Информационные технологи. – 2001. – №3. – С. 27–31. 19. Мукачева Э.А. Рациональный раскрой промышленных материалов. Применение АСУ / Э.А. Мукачева. – М.: Машиностроение, 1984. – 176 с. 20. www.optimalprograms.com/RealCut2d.htm.

УДК 004.021

В.І. Каркульовський, Ю.Р. Дарнобит
Національний університет "Львівська політехніка",
кафедра систем автоматизованого проектування

АЛГОРИТМ РОЗМІЩЕННЯ ФІГУР НА ПЛОЩИНІ З ВИКОРИСТАННЯМ ГРАФІЧНОГО ПРОЦЕСОРА

Розроблено алгоритм розміщення з використанням графічного процесора. Спроектовано та реалізовано систему, яка використовує шейдерний блок графічного процесора для прискорення розрахунків у задачах розміщення.

Ключові слова – алгоритм розміщення, графічний процесор, шейдерний блок, комп'ютерна графіка.

In research it was developed placement algorithm with usage of graphic processor. Also it was designed and realized system that uses shader domain of graphic processor for speeding-up of placement tasks computations.

Keywords – placement algorithm, graphic processor, shader domain, computer graphics.

Вступ

Створення програмних засобів для графічного відображення наукових, технічних, медичних даних і процесів є однією з областей використання комп'ютерної графіки. Дослідникам, аналітикам і багатьом іншим фахівцям доводиться мати справу з великими обсягами інформації або вивчати перебіг процесів великої складності. За чисельного моделювання можуть створюватись файли, що містять велику кількість значень. Технічні реєструючі пристрої також часто накопичують великі обсяги даних, швидше, ніж оператор може їх фізично обробити. Аналіз цих великих масивів даних з метою передбачення подальшої поведінки відповідних процесів і виявлення закономірностей в багатьох випадках не дає змоги ефективно використати людський потенціал. Проте, якщо форму цих даних перетворити за допомогою методів сучасної візуалізації,

то в багатьох випадках стають очевидними тенденції процесів і кореляція величин. При цьому інтеграція в єдиній системі графіки окремих комп'ютерів, мереж, систем візуалізації і відображення інформації створює нові організаційні можливості для взаємодії людини і комп'ютерної техніки.

Процедури обробки зображень у широкому розумінні — це внесення змін або інтерпретація вже існуючих графічних даних, таких як фотографія, відео тощо. Хоча методи, якими користуються у комп'ютерній графіці і при обробці зображень, часто є однаковими, проте ці дві області пов'язані з фундаментально різними підходами. У комп'ютерній графіці за допомогою комп'ютера формується образ. Методи обробки зображень використовуються для покращання якості відеоданих, їх аналізу або розпізнавання тих чи інших візуальних характеристик. У той самий час методи обробки зображень часто застосовуються у комп'ютерній графіці, а методи сучасної візуалізації можуть бути використані для представлення реальних зображень. Для підвищення продуктивності обчислювальних процесів під час розв'язання задач комп'ютерної графіки і візуалізації можна використати графічні прискорювачі. Наприклад, спеціальна алгоритмізація дає змогу отримати істотне зменшення часу розв'язання таких задач, як візуальне моделювання динаміки фізичних процесів різної природи, цифрова обробка зображень, розв'язання систем лінійних алгебраїчних рівнянь, сортування, матрична алгебра, розв'язання систем диференційних рівнянь в часткових похідних, обробка інформації в базах даних тощо.

Враховуючи вказану вище тенденцію інтеграції засобів комп'ютерної графіки і цифрової обробки зображень, істотний інтерес являють собою результати чисельних експериментів, отриманих з одночасним використанням графічного (GPU) і центрального процесора (CPU).

Використання графічного процесора в алгоритмах розміщення

Сьогодні існує багато алгоритмів розміщення та обробки фігур на площині (FastPlace, Mixed-Mode Placement, генетичні алгоритми та ін.) Ці алгоритми використовуються здебільшого для розміщення елементів на друкованих платах чи для задач розкрою. Однак їх неможливо використати при розподілі обчислень з одночасним використанням графічного і центрального процесора. Тому необхідно їх удосконалити.

Універсальні обчислення з використанням графічного прискорювача GPGPU (General Purpose Graphic Processor Unit) [5, 6] інтенсивно розвиваються в останні 4–5 років після появи високопродуктивних відеокарт з можливістю виконання операцій над дійсними числами. Істотна перевага серійних графічних прискорювачів над масовими центральними процесорами середньої вартості пояснюється тим, що модернізація відеокарти полягає у її заміні, а не в оновленні її частин (процесора, мікросхем пам'яті, материнської плати тощо). Тому у виробників графічних прискорювачів менше проблем з сумісністю компонентів і відповідно більше свободи у підвищенні потужності пристроїв. Внаслідок цього виробники відеоприскорювачів випускають нові продукти раз у півроку, а виробники центральних процесорів – раз у два–три роки.

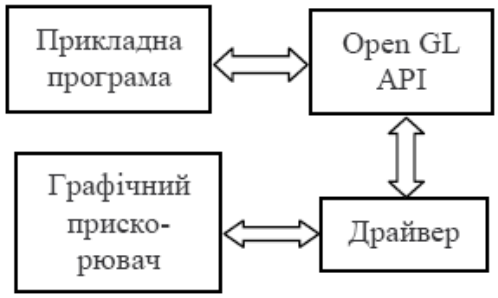

Рис. 1. Функціонування програми з використанням OpenGL

Зазначимо також, що відносне збільшення кількості операцій з плаваючою точкою за секунду від старого покоління до нового для центральних процесорів становить близько 6 гігафлопс у рік, а для графічних прискорювачів – в середньому 25 гігафлопс за рік. Сумісне

використання CPU і GPU вимагає залучення спеціальних бібліотек. Прикладом такої бібліотеки є OpenGL (Open Graphic Library) [3], яка являє собою процедурно-орієнтовану бібліотеку для програмування графіки в реальному масштабі часу з використанням GPU і CPU.

Узагальнену структурну схему, що пояснює функціонування програми з використанням OpenGL API (Application Program Interface), показано на рис. 1.

Бібліотека OpenGL API забезпечує ефективне використання усіх ресурсів сучасних GPU і є посередником між прикладним і графічним прискорювачем, тобто надає програмісту інтерфейс, абстрагований від конкретного графічного прискорювача.

Розробка інформаційної моделі системи

Перш ніж приступити до розробки алгоритмів, необхідно розробити інформаційну модель системи з описом класів і їх взаємозв'язків.

Основним є клас схеми розкрою NestingScheme, який, крім решти параметрів, містить два масиви: перший – масив геометричних фігур, які будуть розміщуватись під час розв'язання задачі розкрою. Другий – масив заготовок, на яких будуть розміщуватись деталі. Обидва масиви побудовані на базі класу List<type>, кожен елемент якого буде посиланням на об'єкт будь-якого типу. Відповідно в масиві деталей кожен елемент буде посиланням на об'єкт деталі Shape, а в масиві заготовок – на об'єкт BoundingBox.

Класс Shape, оскільки він являє собою деталь повинен містити інформацію про її геометрію. Геометрію можна описати як послідовність точок, які треба послідовно з'єднати відрізками. З цього виходить, що деталь може бути описана лише одним способом. Клас NestingScheme є основним. Якраз у ньому проводитиметься розміщення об'єктів класу Shape.

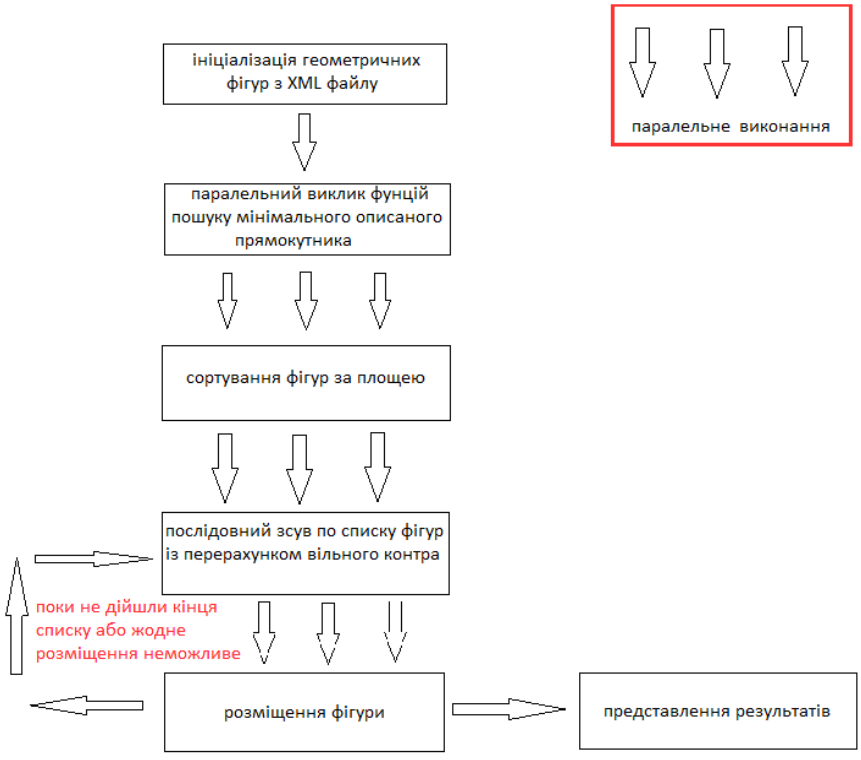

Рис. 2. Логіка розподілу задач між центральним і графічним процесором

Для роботи основному класу потрібно знати зайнятий і вільний контури на кожному кроці розв'язку задачі. Кожен примірник класу Shape може сам знайти своє оптимальне положення на заготовці, що, своєю чергою означає, що він вміє рахувати свою площу і орієнтуватись на площині для отримання описаного прямокутника мінімальної форми. На рис. 2 показано розподілення задач між центральним процесором і графічним процесором. «Паралельне виконання» значить виконання на шейдерному блоці графічного процесора.

Розробка алгоритму

Розв'язок задач фігурного розкрою загалом відбувається у такій послідовності: оператор вводить дані, які містять усю інформацію про геометрію деталей (у нашому випадку ввід відбувається з XML-документа), після того, як ввід було завершено, оператор ініціює запуск розрахунку схеми розкрою. Ця підпрограма проводить розрахунки з пошуку заготовки і розміщення контура деталі всередині заготовки. Далі відбувається оптимізація контура заготовки так, щоб деталь оптимально розміщалась на ній. Потім повертаємо фігуру на потрібний кут для оптимального її вписування у мінімальний прямокутник і проводимо її відображення в бітмап (текстуру). Опісля треба отримати карту кольорів цього бітмапа. Отримання оптимальної текстури є графічною задачею, тому буде виконуватися паралельно для усіх об'єктів у списку. Далі наступає задача сортування текстур за розміром в порядку спадання, що є також графічною задачею. Можливо, не стільки графічною, як задачею, що використовує графічні типи даних. Розміщення проводиться за допомогою використання фізичних законів із динамічним розрахунком геометричного положення із використанням сталої прискорення вільного падіння і розрахунку вектора нормалі, доки оптимальне положення не буде знайдене для кожного рухомого геометричного об'єкта в кадрі. Ефективність полягає у тому, що алгоритм передбачає аналіз розмірів фігури і вільного місця на листі для розкрою за паралельного розміщення об'єктів.

Програмна реалізація

Для реалізації програмного рішення була обрана платформа Microsoft.NET і мова C# [2]. Мова C# є однією з найпростіших і в той самий час однією з найрозвиненіших мов об'єктно-орієнтованого програмування.

Рис. 3. Попіксельна перевірка зіткнення двох геометричних об'єктів. В одному випадку відбувається зіткнення між двома об'єктами. Один із об'єктів підсвічено червоним кольором

Для технології NET існує широка база засобів розробки графічних прикладних програм, взаємодії із графічним апаратним забезпеченням і весь час ведуться роботи з переведення інших технологій на цю платформу. Як середовище програмування вибрано Microsoft Visual Studio 2010.

Створивши список фігур, можна розпочинати їхнє розміщення. Ця задача виконується багатопоточно з використанням графічного процесора. Для цього нам потрібен деякий додатковий фунціонал для передбачення зіткнень [7]. Для більшої точності було обрано попіксельну перевірку зіткнень об'єктів за принципом «кожен з кожним» (рис. 3) і додати для кожного об'єкта в списку фігур так звану гравітаційну сталу і масу (залежить від його площі).

Тепер кожен об'єкт рухатиметься незалежно один від одного до того часу, поки не зіткнеться із собі подібним об'єктом в сцені. Впавши на «дно», об'єкт припиняє рух, але не втрачає своїх фізичних властивостей. Кожен наступний об'єкт вже буде розміщений із врахуванням геометрії попереднього об'єкта і знову ж таки сталої сили тяжіння на усі об'єкти.

Координата X на початку руху фігури вираховується у випадковий спосіб, при цьому фігура повинна бути повністю видима по усій своїй ширині.

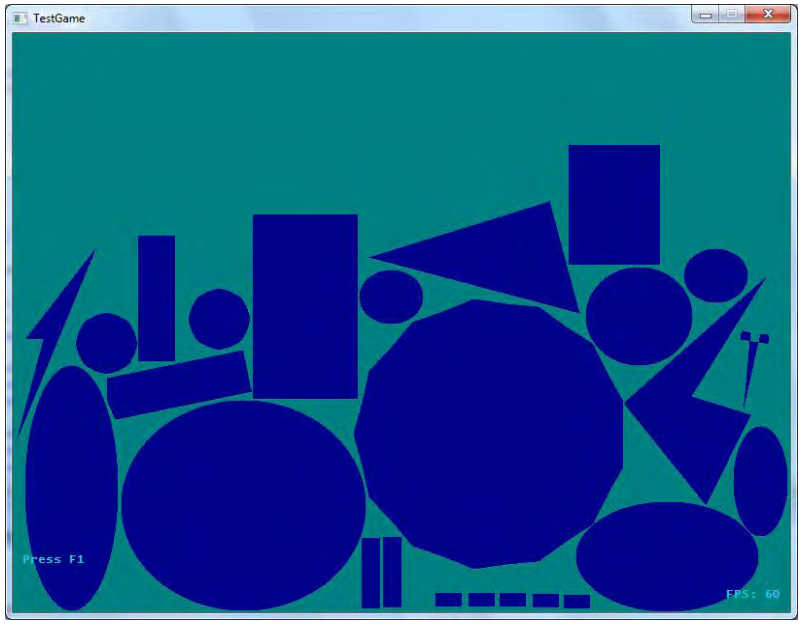

Рис. 4. Приклад результату розміщення. Найменші фігури були розміщені вкінці з врахуванням незайнятого контура заготовки вунапрямку зліва направо і знизу вверх

Алгоритм перевірки зіткнень, використаний під час розміщення, ґрунтується на перетині двох прямокутників, що описують фігуру. Якщо два прямокутники перетинаються, викликається попіксельна перевірка зіткнень. На кожну фігуру діє стала сила тяжіння. Це змодельовано за допомогою вектора руху для кожної фігури, який направлений у додатному напрямку по осі Y. (В комп'ютерній графіці початок координат знаходиться у верхньому лівому куті вікна рендерінга).

Висновки

Розроблено алгоритм розміщення, який використовує для проведення обчислень графічний процесор та працює паралельно з центральним процесором, допомагаючи йому в обчисленнях з плаваючою крапкою. Реалізована система використовує шейдерний блок графічного процесора для прискорення розрахунків у задачах розміщення.

Використання графічного процесора для певного типу математичних задач (обчислення з плаваючою крапкою) дає змогу отримати приріст швидкодії в сотні разів за умови належної оптимізації та відповідної сфери застосування.

1. Стоян Ю.Г., Гиль Н.И. Методы и алгоритмы размещения плоских геометрических объектов. – М., 2000. 2. «Professional C# 2005», Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner, Allen Jones. 3. «The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics», NVIDIA Corporation. 4. «GPU Gems», NVIDIA Corporation. 5. «Professional XNA Programming: Building Games for Xbox 360 and Windows with XNA Game Studio 2.0, 2nd Edition», Benjamin Nitschke. 6. «Microsoft XNA Game Studio 3.0 Unleashed», Microsoft Corporation. 7. «Real-time Collision Detection», Christer Ericson. 8. Accelerator: Using Data Parallelism to Program GPUs for General-Purpose Uses, whitepaper, Microsoft Corporation.