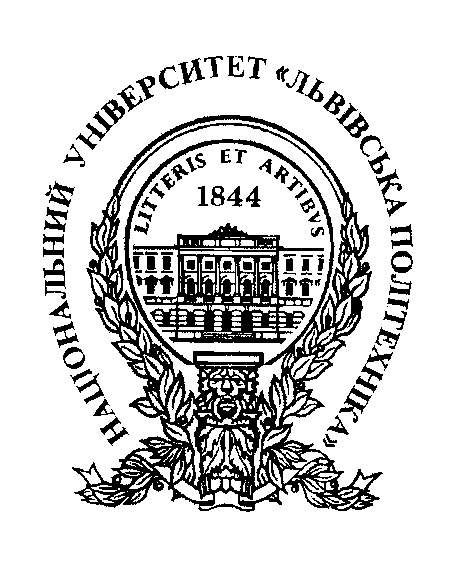
**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Національний університет “Львівська політехніка”**

*Кафедра САПР*



**розрахунково-графічна робота**

**на тему: «Flame gpu: boids partitioning»**

**з курсу «Методи проектування**

**мультиагентних систем»**

Виконав:

ст. групи СПКм-12

Дикий Б. Я.

Прийняв:

Романюк А. Б.

**Львів 2015**

**Мета:** Ознайомитись з основними функціональними можливостями бібліотеки Flame GPU, освоїти сучасні методики та підходи щодо побудови мультиагентних систем за допомогою графічних процесорів (CUDA) та бібліотеки Flame GPU.

1. **Індивідуальне завдання**

Проаналізувати програмний приклад реалізації мультагентної системи «Boids partitioning» на графічному процесорі під управлінням програмно-апартної архітектури паралельних обчислень CUDA.

* 1. **Основні поняття про мультиагентні системи та використання графічних процесорів в мультиагентни системах.**

Мультиагентні системи застосовуються в різних аспектах сучасного життя, таких як, комп'ютерні ігри, кінематографі, менеджменті, продажу, логістиці, математичній статистиці, також теорія иультиагентни систем використовується в складених системах оборони, транспорті, графіці, [геоінформаційних системах](http://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) і багатьох інших. Багатоагентні системи добре зарекомендували себе в сфері мережних і мобільних технологій, для забезпечення автоматичного й динамічного балансу завантаженості, розширюваності й здатності до самовідновлення.

Моделювання на базі агентів (МБА) – нова парадигма моделювання й одна з найбільш захоплюючих в практичному моделюванні з часів винайдення зв’язних баз даних. МБА є новим підходом до моделювання систем, що складаються з автономних, взаємодіючих агентів. МБА має перспективу використання бізнесом у якості систем підтримки рішень. Дехто стверджує, що МБА це третій шлях робити науку крім дедукції та індукції. Розвиток апаратного забезпечення дозволив ріст числа агентно-базованих програм серед множини інших. Програми розвинулись від моделювання поведінки агента на складах й каналах постачання, до прогнозування поширення епідемій і загрози біологічної війни, від моделювання поведінки клієнта до моделювання падіння древніх цивілізацій і т.д.

**Мультиагентні системи** - це система, утворена декількома взаємодіючими [інтелектуальними агентами](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%B0%D0%B3%D0%B5%D0%BD%D1%82). Мультиагентні системи  можуть бути використані для розв'язання таких проблем, які складно або неможливо вирішити за допомогою одного агента або [монолітної системи](http://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%BD%D0%BE%D0%BB%D1%96%D1%82%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0&action=edit&redlink=1). Прикладами таких завдань є онлайн-торгівля, ліквідація надзвичайних ситуацій і моделювання соціальних структур.

**Графічний процесор** - окремий пристрій персонального комп'ютера або ігрової приставки, виконує графічний рендеринг. Сучасні графічні процесори дуже ефективно обробляють і зображують комп'ютерну графіку, завдяки спеціалізованій конвеєрній архітектурі вони набагато ефективніші в обробці графічної інформації, ніж типовий центральний процесор.

Графічний процесор в сучасних [відеоадаптерах](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%B5%D0%BE%D0%B0%D0%B4%D0%B0%D0%BF%D1%82%D0%B5%D1%80) використовується в якості прискорювача [тривимірної графіки](http://uk.wikipedia.org/wiki/3D_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0), але в деяких випадках його можна використовувати і для обрахунків ([GPGPU](http://uk.wikipedia.org/wiki/GPGPU)). Обрахунковими особливостями в порівнянні із [CPU](http://uk.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80) є:

* архітектура, максимально націлена на збільшення швидкості обчислень [текстур](http://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D1%83%D1%80%D0%B0_(%D1%82%D1%80%D0%B8%D0%B2%D0%B8%D0%BC%D1%96%D1%80%D0%BD%D0%B0_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0)) та складних графічних об'єктів;
* обмежений список команд;
* оптимізований процес паралельного виконання.

Моделювання мультиагентних систем - це методика комп'ютерного моделювання складних систем, що взаємодіють через специфікації поведінки ряду автономних індивідів, що діють одночасно. Це підхід "знизу вгору", на відміну від підходу «згори вниз», в якому моделювання поведінки системи відбувається через динамічні математичні рівняння. Незважаючи на очевидну необхідність паралелізму, часто мультиагентні системи використовують послідовну форму алгоритмів маніпулювання мобільних дискретних агентів. Такий підхід має серйозні недоліки, такі як, обмеження при розміщенні агентів та швидкості в цілому.

Для усунення вище описаних недоліків, для побудови ефективних і сучасних мультиагентних систем, які потребують великої кількості агентів та обчислень в цілому, використовують потужності графічного процесора.

* 1. **Доцільність використання графічного процесора при побудові мультиагентних систем**

При побудові мультиагентних систем, необхідно звертати уваги на відмінності обробки запитів CPU i GPU. Існують такі основні відмінності у роботі універсального процесора та графічного процесора:

* Ядра CPU створені для виконання одного потоку послідовних інструкцій з максимальною продуктивністю, а GPU проектуються для швидкого виконання великої кількості паралельно виконуваних потоків інструкцій.
* Універсальні процесори оптимізовані для досягнення високої продуктивності єдиного потоку команд, обробного і цілі числа і числа з плаваючою крапкою. При цьому доступ до пам'яті випадковий.
* Принцип доступу до пам'яті. У СPU він пов'язаний і легко передбачуваний. І графічному процесорі, на відміну від універсальних процесорів, просто не потрібна кеш-пам'ять великого розміру.
* В універсальних процесорах велика кількість транзисторів і площа чіпа йдуть на буфери команд, апаратне пророкування розгалуження і величезні обсяги начіповой кеш-пам'яті. Всі ці апаратні блоки потрібні для прискорення виконання нечисленних потоків команд. Відеочіпи витрачають транзистори на масиви виконавчих блоків, керуючі потоками блоки, пам'ять, що розділяється невеликого обсягу і контролери пам'яті на кілька каналів.

Графічні процесори розраховані на багатопоточну роботу в паралельному режимі. Це зумовлює низку переваг, але доцільність використання такого підходу, буде виправдана лише в випадку великої кількості агентів та повідомлень між ними. За маленької кількості агентів, доцільнішим є використання центрального процесору, що може дати кращий результат, ніж виконання за допомогою графічного процесора. Це зумовлене архітектурою та специфікою обробки запитів процесором і доступу до пам’яті.

* 1. **Принципи закладені в бібліотеку Flame GPU**

Технічно фреймворк Flame GPU - не симулятор, а  шаблон на основі середовища моделювання, яке відображає офіційні описи агентів в коді середовища. Формальне представлення агента базується на концепції комунікації X-машини (яка є розширенням Finite State-машини, яка включає в себе пам'ять). Незважаючи на те, що  Х-машина має дуже формальне визначення, агенти Х-машини можуть розглядати стан машин, які здатні спілкуватися за допомогою повідомлень, які зберігаються в загально доступних списках повідомлень. Функціональність агента подається як набір функцій переходів, які переміщають агентів з одного внутрішнього стану до іншого. При зміні стану, агенти оновлюють свою внутрішню пам'ять в залежності від повідомлень, які можуть бути використані або в якості входу (шляхом перебору списків повідомлень), або як вихід (де інформація може бути передана в списках повідомлень для прочитання іншими агентами). З цією метою Flame GPU використовує сценарії функцій агента, де сценарій визначається в ряді файлів функцій агента. Симулятори задаються за допомогою мови розмітки (XMML), яка є синтаксисом XML  з відповідними схемами, регулюючими зміст. Як правило, XMML файл моделі складається з визначеної кількості агентів Х-машини, ряду деяких типів повідомлень (кожен з яких має загальнодоступний список повідомлень) і з набору шарів  моделювання, які визначають порядок виконання функцій агента для однієї ітерації моделювання. Протягом моделювання дані агента є стійкими, однак інформаційне повідомлення (і, зокрема списки повідомлень) постійні тільки протягом життєвого циклу однієї ітерації. Це дозволяє механізму агентів багаторазово взаємодіяти і формувати групову поведінку.

* 1. **Кроки необхідні для запуску чи реалізації мультиагентної системи за допомогою Flame GPU. Компіляція в Visual Studio.**

Flame GPU SDK орієнтована на CUDA 4.з використанням Visual Studio починаючи з версії 2008. Експорт в середовища на операційній системі Linux повинне бути відносно простим. Перевагою використання Visual Studio є наявність візуального редактору XML, що включає підтримку перевірки XML-тегів і автозаповнення, за допомогою якого створення моделі XML, відбувається досить просто.

*Зміна налаштувань побудови проекту*

Приклади бібліотеки Flame GPU і файли шаблонів проекту конфігуровані для 32 і 64-бітних ОС Windows, які можуть бути змінені за допомогою меню "Solution Platforms". Для кожної платформи, в проекті містяться чотири конфігураційних файли для відлагодження (debug) і випуску робочих версій (release). Конфігураційні файли візуалізації дозволяють заздалегідь вказати процесор та макрос візуалізації, який використовується в ряді попередніх умовних процесорних функціях моделювання очікуваних аргументів.

*Структура файлів проекту в Visual Studio*

В прикладах Flame GPU віртуальні папки організовані наступним чином:

* Flame GPU - складається з папки, що містить XML-схема.
* Flame Model - містить файл моделі XML і файл функції агента (зазвичай називається "functions.c").
* Dynamic Code - містить динамічно згенерований код Flame GPU моделювання. Цей код буде переписаний кожен раз при збірці проекту.
* Additional Source Code - ця папка повинна містити файли заголовків. За замовчуванням шаблон проекту Flame GPU визначає один ”visualisation.h” файл у цій папці, який може бути змінений.

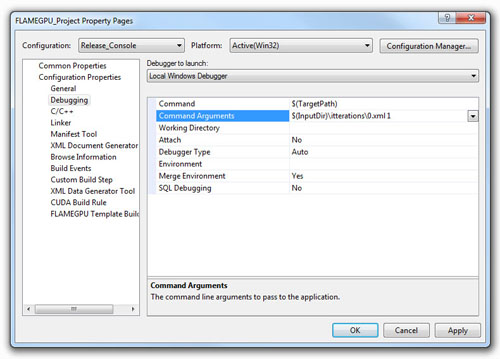
*Процес збірки проекту*

Процес збірки проекту в Visual Studio складається з ряду етапів:

1. Інструмент для збірки, який генерує динамічний код моделювання з шаблонів.
2. Після цього генерується код, за правилами збірки CUDA, та компілюється у вихідні файли, використовуючи компілятор Nvidias NVCC CUDA.
3. Нарешті, С чи С++ вихідні файли компілюються за допомогою компілятора MSVC, а потім пов'язуються з об’єктами файлів CUDA для генерації виконавчого файлу.

*Аргументи конфігурацій запуску проекту*

Для того, щоб встановити аргументи запуску необхідно зайти в пункт меню "Проект"->"Властивості" -> "Аргументи запуску". Кожна конфігурація має свій власний набір аргументів запуску.



*Рис. 1. Вікно задання аргументів запуску проекта.*

* 1. **X-Machines**

У файлі XMML повинен бути принаймні один елемент xagent, який повинен визначити принаймні одного xagent. Xagent є агентом представлення X-Machines і складається з імені, опису , набору функцій, внутрішньої пам'яті і безлічі станів.  Для FLAME GPU потрібно дві додаткові інформації (які не потрібні в оригінальній специфікації XMML), тип і розмір буфера. Тип елемента відноситься до типу агента по відношенню до його зв'язку з його просторовим середовищем. Тип агент може бути дискретним або безперервним. Вся пам'ять попередньо виділяється на GPU розмір буфера потрібно представляти максимально можливий розмір населення агента. Тобто максимальне число випадків X-Machines агента у форматі, описаному в моделі XMML. Це не є недоліком продуктивності використання великого розміру буфера, це відповідальність користувача, що GPU містить достатньо пам'яті для підтримки великих популяцій агентів. Рекомендується розмір буфера  завжди степінь двох чисел (тобто 1024, 2048, 4096, 16384 і т.д.),  швидше за все, округлиться до одного під час моделювання. Для дискретних агентів розмір буферу в даний час обмежується ступенем двох чисел. Якщо в будь-який момент у процесі моделювання перевищить розмір буферу то користувач буде про це попереджений.

*Agent Memory* складається з ряду змінних (принаймні, одного), які використовуватимуться для зберігання інформації. Змінна агента повиненна мати унікальне ім'я і може бути типу int, float or double. В даний час Agent Memory підтримує тільки використання окремих областей пам'яті (тобто не статичні або динамічні масиви) і значення за замовчуванням завжди 0, якщо значення не вказано в XML стану входів файл.

*Agent State* визначаються у вигляді списку елементів (Q в X-Machine формального визначення) з унікальним ім'ям. Моделювання в межах FLAMEGPU може тривати нескінченно (або на певну кількість ітерацій), термінальні стани не визначені. Початковий стан (q0) визначається в елементі InitialState і повинен відповідати існуючій і єдиній політиці імен зі списку.

**Результат**

Нижче приведені результати роботи мультиагентної системи boids partitioning на різних етапах роботи:

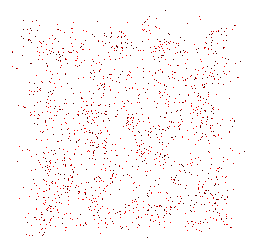


Рис. 2. Виконання МА Boids Partitioning. Початковий етап.

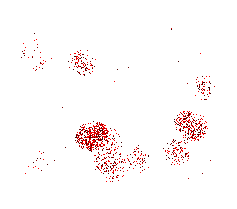


Рис. 3. Виконання МА Boids Partitioning. Фінальний етап.

**Висновок**

В даній розрахунково-графічній роботі, я ознайомився з основними функціональними можливостями бібліотекою Flame GPU. Також було проведено аналіз існуючих мультиагентних систем реалізованих за допомогою цієї бібліотеки. Було вивчено сучасні підходи щодо побудови мультиагентних систем та засвоєно правило вибору процесора для опрацювання поведінок агентів.

На основі мультигентної системи Boids Partitioning, було проведено дослідження роботи мультиагентних систем з опрацюванням графічним процесором.