Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**ЗВІТ**

**про проходження виробничої практики**

**Студента** 4 курсу групи ІО-33

**Спеціальності** 6.050102 «Комп’ютерна інженерія»

Косовіча Веліміра

**База практики:** «ПТС ЮА Сервісез»

Керівник практики:

Селіванов В.Л.

Київ 2017

**ЗМІСТ**

Вступ

1. Огляд сучасних фізичних рушіїв

2. Створення примітивної системи колізій простих об’єктів

3. Оптимізація системи

Висновки.

Література

ВСТУП

У кожному вузі України передбачено проходження студентами виробничої практики, яка є обов’язковим складовим елементом підготовки будь-якого спеціаліста. Під час проходження практики студент закріплює здобуті знання і навички, отримуючи практичний досвід роботи в реальних економічних умовах.

Студент може мати досконалі теоретичні знання, але може розгубитися в ситуації, коли виникне необхідність прийняти рішення в реальних економічних умовах. Тому необхідно забезпечувати студентам змогу пройти виробничу практику ще під час навчання у вузі, для того, щоб окрім теоретичних відомостей отримати й практичні навички роботи на підприємстві.

Виробнича практика передбачає отримання необхідного обсягу практичних знань та умінь відповідно до кваліфікаційного рівня інженера-системотехніка.

Метою виробничої практики для студентів спеціальності 6.050102 є оволодіння студентами сучасними методами ефективного застосування комп'ютерних та мережних технологій для вирішення задач комплексної автоматизації підприємств і установ, вивчення практичних рішень в галузі організації та побудови регіональних, корпоративних та локальних комп'ютерних мереж, розподілених комп'ютерних систем, дослідження характеристик, виконуваних функцій, етапів розробки, тестування та впровадження використовуваних на об'єктах практики універсального та спеціалізованого апаратного та програмного забезпечення комп'ютерних систем та мереж.

1. Огляд сучасних фізичних рушіїв.

**Фізичний рушій –**  [програмний рушій](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D1%80%D1%83%D1%88%D1%96%D0%B9" \o "Програмний рушій), що робить симуляцію фізичних законів реального світу у світі віртуальному з тим або іншим ступенем апроксимації. Найчастіше фізичні рушії використовуються не як окремі самостійні програмні продукти, а як складені компоненти (підпрограми) інших програм[1].

Сучасні фізичні рушії симулюють не всі фізичні закони реального світу, а лише деякі, причому із часом і прогресом у галузі інформаційних технологій і обчислювальної техніки список «підтримуваних» законів збільшується. На початок[2010](https://uk.wikipedia.org/wiki/2010) року фізичні рушії можуть симулювати такі фізичні явища й стани:

* динаміка [абсолютно твердого тіла](https://uk.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D0%BE%D0%BB%D1%8E%D1%82%D0%BD%D0%BE_%D1%82%D0%B2%D0%B5%D1%80%D0%B4%D0%B5_%D1%82%D1%96%D0%BB%D0%BE);
* динаміка [деформівного](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%84%D0%BE%D1%80%D0%BC%D1%96%D0%B2%D0%BD%D1%96%D1%81%D1%82%D1%8C" \o "Деформівність) тіла;
* динаміка рідин;
* динаміка газів;
* поведінка тканин;
* поведінка мотузок (тросів, канатів тощо).

На сьогоднішній день існує досить багато таких систем, як безкоштовних та з відкритим виконавчим кодом, так і комерційні та потужні. Деякі з них : [nVidia PhysX](https://uk.wikipedia.org/wiki/PhysX_(%D1%80%D1%83%D1%88%D1%96%D0%B9)),  [Havok](https://uk.wikipedia.org/wiki/Havok" \o "Havok), [Bullet Physics Library](https://uk.wikipedia.org/w/index.php?title=Bullet_Physics_Library&action=edit&redlink=1), Box2D, Source.

Фізичний рушій дозволяє створити деякий віртуальний простір, який можна наповнити тілами (віртуальними статичними й динамічними об'єктами), і вказати для нього якісь загальні закони взаємодії тіл і середовища, тією, чи іншою мірою наближені до фізичних, задаючи при цьому характер і ступінь взаємодій (імпульси, сили тощо). Властиво розрахунок взаємодії тіл рушій і бере на себе. Коли простого набору об'єктів, що взаємодіють за певними законами у віртуальному просторі, недостатньо в силу неповного наближення фізичної моделі до реального світу, можливо додавати (до тіл) зв'язки. Розраховуючи взаємодію тіл між собою й із середовищем, фізичний рушій наближає фізичну модель одержуваної системи до реального світу, передаючи уточнені геометричні дані засобові відображення.

1. Створення примітивної системи колізій простих об’єктів

Дану систему має включати кожний фізичний рушій. Вона моделює поведінку об’єктів при дотику з іншими об’єктами. В подальшому об’єкт означатиме тверде тіло, а швидкість об’єкта буде представляти вектор, що буде показувати напрям та містити модуль цієї швидкості.

Система включає головний контейнер, що буде моделювати простір, і буде зберігати всі тіла. Саме цей контейнер буде рахувати поведінку тіл при дотику, та змінювати їхню швидкість в залежності від обставин.

Також система має представляти простий опис твердих тіл. Для примітивів були вибрані ліній та круг. Використовуючи ці фігури можна створювати більш складні тіла.

Для пошуку дотику було вирішено використовувати систему raycast. В перекладі це означає трасування променів. Тобто з однієї точки об’єкту пускається промінь в напрямку руху об’єкту. Якщо він зустрічає якийсь об’єкт, він повідомляє про це контейнер, що в свою чергу дивиться на відстань променя, та вирішує чи змінювати швидкість об’єкта чи ні.

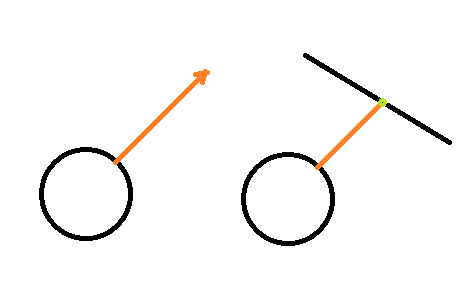


Рис. 1.1. Приклад роботу трасування променя.

Також система має містити модуль корекції положення. У випадках коли контейнер прийняв неправильне рішення, і не змінив швидкість, або зробив це занадто пізно. Нижче на рисунку показано, як працює ця система. Коли контейнер помічає, що два тіла пересіклись, то він відсуває менше динамічне тіло на відстань дотику(проекція на вектор нормалі для цього тіла перетину тіл) вздовж нормалі для цього тіла.

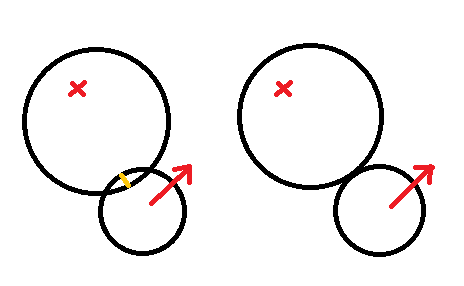


Рис. 1. 2. Приклад роботи системи корекції положення тіл

На виході отримали систему, що може моделювати рух твердих тіл в обмеженому 2D просторі з використанням приближеними до реальних умов формулами, що показують доволі непогані результати.

1. Оптимізація системи

Головними проблемами були використання великої кількості малих об’єктів при обрахунках. Та великі затрати на пошук дотиків.

Першу проблему вдалось вирішити за допомогою паттерна Pool, який дозволяє використовувати одні і ті самі об’єкти багаторазово. При цьому відпадають затрати на створення об’єктів та видалення їх з пам’яті. За рахунок цього за один цикл моделювання, було скорочено створення дрібних математичних об’єктів з 12500 до 250 в середньому, що я вважаю досить непогано.

Приклад реалізації абстрактної моделі пула об’єктів:

public abstract class ObjectPool<T> {

private long expirationTime;

private Hashtable<T, Long> locked, unlocked;

public ObjectPool() {

expirationTime = 30000;

locked = new Hashtable<T, Long>();

unlocked = new Hashtable<T, Long>();

}

protected abstract T create();

public abstract boolean validate(T o);

public abstract void expire(T o);

public synchronized T checkOut() {

long now = System.currentTimeMillis();

T t;

if (unlocked.size() > 0) {

Enumeration<T> e = unlocked.keys();

while (e.hasMoreElements()) {

t = e.nextElement();

if ((now - unlocked.get(t)) > expirationTime) {

unlocked.remove(t);

expire(t);

t = null;

} else {

if (validate(t)) {

unlocked.remove(t);

locked.put(t, now);

return (t);

} else {

unlocked.remove(t);

expire(t);

t = null;

}

}

}

}

t = create();

locked.put(t, now);

return (t);

}

public synchronized void checkIn(T t) {

locked.remove(t);

unlocked.put(t, System.currentTimeMillis());

}

}

Друга проблема була вирішена розширенням контейнера за допомогою окта дерева, тільки на 2 площинах.

Октодерево - [дерево](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)), у якому кожна вершина має вісім дітей. Вісімкові дерева найчастіше використовуються, щоб поділити тривимірний простір рекурсивним розбиттям на [октанти](https://uk.wikipedia.org/wiki/%D0%9E%D0%BA%D1%82%D0%B0%D0%BD%D1%82" \o "Октант).

В моєю випадку потрібно було використовувати квадратне дерево, яке забезпечувало цей функціонал на площині.

Далі неведено основні класи реалізації:

public class QTree implements BroadPhase {  
   private final QTreeNode node;  
     
   // define a quadtree extends as width and height, define quadtree depth.  
   public QTree(final float worldExtends, int worldDepth) {  
      node = new QTreeNode(0,0,worldExtends, worldDepth);  
   }  
  
   // insert a GameObject at the quadtree  
   public void insertObject(final GameObject obj) {  
      node.insertObject(obj, obj.circleCollider);  
   }  
     
   // clean the quadtree  
   public void clean() {  
      node.clean();  
   }  
}

public class QTreeNode {  
   private final int currDepth; // the current depth of this node  
   private final Vector center; // the center of this node  
   private final QTreeNode[] nodes; // the child nodes  
     
   private final ArrayList<GameObject> objects; // the objects stored at this node  
     
   public QTreeNode(float centerX, float centerY, float halfWidth, int stopDepth) {  
      this.currDepth = stopDepth;  
        
      // set Vector to current x-y-z values  
      this.center = new Vector(centerX, centerY, 0.0f);   
      this.objects = new ArrayList<GameObject>();  
      if (stopDepth > 0) {  
         // create 4 child nodes as long as depth is still greater than 0  
         this.nodes = new QTreeNode[4];  
           
         // halve child nodes size  
         float step = halfWidth \* 0.5f;  
           
         // loop through and create new child nodes  
         for (int i = 0; i < 4; i++) {  
              
            // compute the offsets of the child nodes  
            offsetX = (((i & 1) == 0) ? step : -step);  
            offsetY = (((i & 2) == 0) ? step : -step);  
              
            nodes[i] = new QTreeNode(centerX + offsetX, centerY + offsetY, step, stopDepth - 1);  
         }     
      }  
      else {  
         this.nodes = null;  
      }  
   }  
     
   public void insertObject(final GameObject obj, final Circle collider) {  
      int index = 0; // get child node index as 0 initially  
      boolean straddle = false; // set straddling to false  
      float delta;  
        
      // get the raw arrays, makes it easier to run these in a loop  
      final float[] objPos = collider.center.vec;  
      final float[] nodePos = center.vec;  
        
      for (int i = 0; i < 2; i++) {  
         // compute the delta, nodePos Vector index - objPos Vector  
         delta = nodePos[i] - objPos[i];  
           
         // if the absolute of delta is less than or equal to radius object straddling, break  
         if (Math.abs(delta) <= collider.radius) {  
            straddle = true;  
            break;  
         }  
           
         // compute the index to isnert to child node  
         if (delta > 0.0f) {  
            index |= (1 << i);  
         }  
      }  
        
      if (!straddle && currDepth > 0) {  
         // not straddling, insert to child at index  
         nodes[index].insertObject(obj, collider);  
      }  
      else {  
         // straddling, insert to this node  
         objects.insert(obj);  
      }  
   }  
     
   public void clean() {  
      objects.clear();  
        
      // clean children if available  
      if (currDepth > 0) {  
         for (int i = 0; i < 4; i++) {  
            nodes[i].clean();  
         }  
      }  
   }  
}

Після інтегрування цього дерева в систему, час для пошуку колізії вдалось скоротити в 3 рази в середньому.

Висновки

­­Виробнича практика є джерелом накопичення практичного досвіду роботи по вибраній спеціальності та закріплення теоретичних знань. Вона відіграє важливу роль у підготовці спеціалістів високого класу. Адже важливо не лише мати глибокі теоретичні знання зі спеціальності, а й уміння застосовувати їх на практиці під час виконання реальних виробничих задач.

Під час проходження практиці в «ПТС ЮА Сервісез» було отримано величезний досвід в розробці додатків для мобільних пристроїв, та написання своїх систем моделювання фізичних явищ. Було розглянуто не тільки основи розробки програмного забезпечення, а й досвід розробки в великій команді людей, які працюють над одним проектом. Що я вважаю, є не менш важливим фактором, який дозволяє відчути практика.

Список використаної літератури:

1. <https://uk.wikipedia.org/wiki/Фізичний_рішій>
2. <https://unity3d.com/learn/tutorials/modules/beginner/physics/raycasting>
3. <https://ru.wikipedia.org/wiki/Октодерево>
4. <https://sourcemaking.com/design_patterns/object_pool>
5. https://www.java-gaming.org/index.php?topic=27334.0