Юнити для гемдев; з редактором; керований код; ООП; C#

розробляти додатки на Unity3D можна використовуючи лише керований код

#### Взаємодія з бібліотекою драйверу пристрою

Реалізація взаємодії графічного модуля з приладом, перш за все потребує взаємодії з його драйвером, який, в свою чергу, організує реальний обмін даними з апаратною частиною. Драйвер, який описано раніше, розроблений у вигляді бібліотеки з використанням мови програмування C++ на основі некерованого коду. Це може викликати деякі труднощі при взаємодії з керованим кодом мови C#.

На щастя Unity3D має засоби, що дозводять взаємодіяти з такою бібліотекою, через механізм так званих «Native plugins». Для цього необхідно додати файл бібліотеки (.dll) в проект, як звичайний файл ресурсів, але в особливу папку Plugins в корені проекту, оскільки там відбувається пошук всіх файлів бібліотек.

Щоб прив’язати метод який може бути викликаний з керованого коду до функції з бібліотеки обхідно об’явити його як зовнішній статичний (static extern), тобто для виклику з бібліотеки функції void Test(), необхідно записати так: «*public static extern void Test();*». А щоб вказати з якої саме бібліотеки буде викликана функція потрібно використати спеціальний атрибут DllImport зі строковим параметром — ім’ям бібліотеки. [<http://www.alanzucconi.com/2015/10/11/how-to-write-native-plugins-for-unity/>]

Цей спосіб використання бібліотеки дуже легкий і, як стверджують розробники, він працює на всіх платформах, що підтримуються. Щоб її підключити та використовувати функції необхідно всього дві стрічки коду на кожну, а додати до проекту можна простим перетягуванням в папку. Всю іншу роботу система зробить сама. Однак він має один досить суттєвий недолік, який полягає в тому що завантажена одного разу бібліотека вже не буде вивантажена аж до перезавантаження редактора Unity. Тобто вона буде зайнята, а файл захищено від модифікації. Іншими словами, щоб перебудувати бібліотеку необхідно перезапустити редактор.

Зважаючи на те, що перезавантаження редактора займає кілька хвилин, використання цього способу при розробці, коли перебудовувати бібліотеку доводиться постійно, є практично неможливим, навіть з усіма його перевагами.

Альтернативою є динамічне завантаження бібліотеки засобами операційної системи Windows, за допомогою системних функцій LoadLibrary, GetProcAddress та FreeLibrary, які були детально описані у попередньому підрозділі разом із алгоритмом завантаження.

Ці функції завантажуються за допомогою першого способу зі системної бібліотеки «kernel32.dll». Зважаючи на те, що kernel32.dll є частиною ядра операційної системи вона не буде перебудована, тому немає необхідності її вивантажувати, а значить використання першого способу є цілком виправданим.[<https://blogs.msdn.microsoft.com/jonathanswift/2006/10/03/dynamically-calling-an-unmanaged-dll-from-net-c/>]

Завантаження бібліотеки відбувається так само як і в мові C++, за винятком того, що замість вказівників на об’єкти використовується спеціальний тип IntPtr, який по суті є вказівником на область некерованої пам’яті.

Враховуючи це сигнатури функцій, як і перелік кроків, що необхідно виконати буде трохи змінений. Функція LoadLibrary приймає ім’я бібліотеки яку буде завантажено та повертає вказівник у форматі IntPtr. При відсутності вказаного файлу раніше функція порожня значення (NULL), однак згідно з правилами мови програмування змінній типу IntPtr не може присвоєне це значення. Тому для перевірки успішності операції необхідно порівнювати результат зі спеціальним значенням IntPtr.Zero, яке є еквівалентом порожнього вказівника.[<https://msdn.microsoft.com/ru-ru/library/system.intptr.zero(v=vs.110).aspx>]

Функція GetProcAddress, що використовується для пошуку адреси функції отримує на вхід вказівник на завантажену бібліотеку (IntPtr) та строковий параметр з ім’ям функції, для завантаження. Однак в мові C# не можна викликати функцію через IntPtr вказівник. Для цього передбачений спеціальний тип делегат (Delegate) який може містити адресу методів та дозволяє їх викликати. Щоб конвертувати IntPtr вказівник в делегат слід використати метод Marshal.GetDelegateForFunctionPointer, який приймає вказівник та тип (об’єкт класу Type) делегату та повертає його, щойно створений об’єкт.

Функція вивантаження (FreeLibrary) змінила лише тип параметра-вказівника, а в іншому залишилась без змін.[<http://answers.unity3d.com/questions/10216/unload-a-plugin.html>] [<http://answers.unity3d.com/questions/293867/easier-way-to-handle-unloading-dlls.html>]

Залишилось згадати лише про одну особливість такого використання бібліотеки, яка полягає в тому, що абсолютний шлях до неї не повинен містити кириличних символів. В іншому випадку можуть виникнути проблеми з кодуванням і завантаження може не відбутись.

#### Обмін даними між драйвером та модулем відображення

При вимірюванні характеристик виникає необхідність передачі значних об’ємів даних. Механізм передачі потребує окремого розгляду оскільки така передача може зайняти багато часу, що неодмінно позначиться на швидкості відгуку всієї програми. Річ у тім, що графічна частина системи працює в одному потоці, тобто всі дії виконуються по черзі в так званому головному циклі, а якщо якась операція триватиме значний час (навіть секундна затримка буде досить помітною) то програма весь цей час не реагуватиме на дії користувача, більше того буде схоже, що вона «зависла».

Не зважаючи на те, що найбільші затрати часу спричиняє саме передача даних між апаратною частиною та комп’ютером, даний підрозділ присвячений обміну даними між керованим кодом скриптів Unity на мові C# і некерованим кодом бібліотеки, що написана з використанням мови C++.

Як уже сказано найпростіший варіант — виклик функції з бібліотеки, що очікує отримання даний від пристрою та повертає весь масив. У випадку якщо всі дані ще не прийнято функція може повернути порожнє значення (NULL). Він має багато недоліків, яких можна віднести утруднення контролю пам’яті на гранці між керованим та некерованим кодом, особливо якщо взяти до уваги те, що у некерованому коді функція може повернути масив лише у вигляді вказівника на його перший елемент, а в керованому коді робота з вказівниками має деякі складнощі. Другий недолік — складність встановлення розмірів отриманих даних.

Щоб пришвидшити відгук програми при передачі інформації можна розбити масив даних на менші частини. Однак і це не вирішує граничних проблем з виділенням та звільненням некерованої пам’яті та потребує окремої передачі розміру переданої частини масиву.

З огляду на недоліки було прийнято рішення відмовитись від прямого повернення масивів даних з некерованого коду.

Щоб уникнути очікування прийняття всіх даних можна використати механізм обробки подій, що заснований на застосуванні функцій зворотного виклику — тепер функція керованого коду буде викликана з некерованого.

При відправленні запиту на отримання масиву даних до некерованого коду драйверу передається вказівник на функцію, що буде викликана після завершення отримання даних від апаратної частини. Іншими словами відбудеться подія звершення прийняття і всі модулі що «підписані» на неї отримають (в даному випадку це лише графічний модуль) повідомлення і зможуть обробити цю подію. Проте, прийняття даних відбувається в додатковому потоці, і очевидно, функція-обробник буде викликана в тому ж таки додатковому потоці. Тобто окрім синхронізації потоків на рівні драйвера необхідна додаткова синхронізація на рівні керованого коду.[<http://www.realcoding.net/articles/glava-17-vzaimodeistvie-s-neupravlyaemym-kodom.html>]

Найбільш оптимальне рішення полягає спільному використанні пам’яті до якої мають доступ обидва модулі програми. Некерований модуль не може отримати доступ до керованої пам’яті. А в мові C# є всі необхідні засоби для роботи з некерованою пам’яттю яку використовує C++.

Спільно використовувати можна будь-яку структуру даних, що може включати в себе масив з даними, його розміри та, за необхідності, іншу інформацію.

Для реалізації такого обміну необхідно створити однакові структури на C++ та C#. У випадку останньої використовуються спеціальні атрибути, що допоможуть правильно розташувати поля керованої структури. В некерованій бібліотеці створити функцію що отримує вказівник на цю структуру(SetBuffer), зберігає його та записує туди дані по мірі їх отримання від приладу та функцію, що показує чи завершена передача даних (IsReseived), вона поверне логічне значення FALSE доки не буде отримано всі дані, а потім поверне TRUE.

Для початку процедури отримання даних потрібно виконати наступні кроки. Спершу потрібно створити об’єкт керованої структури. Потім виділити некеровану пам’ять під структуру та скопіювати туди щойно створений об’єкт цієї структури. Після копіювання буде отриманий вказівник на цю область пам’яті, який треба передати до бібліотеки викликом відповідної функції, що також означатиме початок очікування передачі.

Далі на кожній ітерації основного циклу необхідно перевіряти чи вже отримано всі дані. Після того як це станеться можна отримати новий об’єкт керованої структури з цієї області некерованої пам’яті та вивільнити її.

Основною перевагою цього варіанту передачі є те, що необхідна синхронізація потоків лише на ріні драйверу. Іншою превагою є необхідність лише двох функцій для організації такої передачі та прозора робота з пам’яттю.[<https://habrahabr.ru/post/84076/>]

Полігони

Карта висот, порохня

#### Вибір способу побудови поверхні

Відображення графіка функції двох змінних

Графік функції двох змінних відображається у вигляді поверхні.

Як і більшість сучасних засобів відображення тривимірної графіки, Unity зазвичай використовує готові тривимірні моделі, які попередньо завантажені з файлу.

З огляду на це для побудови поверхні можна було б використати заздалегідь заготовлені полігони у вигляді окремих моделей. Проте, перепад висоти між сусідніми точками може сильно відрізнятись, а отже різними будуть розміри та форма трикутників.

Впоратись з цією ситуацією можна двома шляхами. Або підготувати багато полігонів для всіх можливих випадків, що неодмінно призведе до невиправданих витрат пам’яті. Або написати досить складний алгоритм масштабування кожного полігону в залежності від необхідних розмірів, що неодмінно призведе до появи помилок. З цього можна зробити висновок, що дане рішення не є оптимальним.

Знаючи, що карти висот часто використовуються для моделювання ландшафтів місцевості, розробники включили в стандартну поставки спеціальний інструмент Terrain, що дозволяє динамічно створювати ландшафти прямо редакторі, або навіть під час роботи програми. Він містить досить розвинений функціонал наприклад для пошуку точок перетину інших об’єктів з поверхнею ландшафту, або динамічного нанесення текстур. Це все може призвести до збільшення затрат ресурсів.

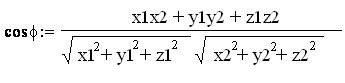
З огляду на все це найбільш оптимальним способом для відображення поверхні є використання низькоірвневих засобів побудови графічних примітивів, що майже напряму використовують засоби графічних адаптерів.

#### Побудови тривимірного примітиву

Звичайний опис тривимірних моделей складається зі структури даних до якої входять наступні масиви даних.

**Масив вершин** містить координати точок з координатами (x, y, z), які є вершинами полігонів-трикутників.

**Масив нормалей**. Нормалі зазвичай використовуються для розрахунку освітленості поверхні — чим більший кут між вектором нормалі та вектором освітлення тим більш яскраво буде освітлений полігон. Якщо є вектори з координатами (x1, y1, z1) та (x2, y2, z2), то кут між ними може бути знайдений за наступною формулою:



Скалярний добуток цих векторів розділений на добуток їх довжин. Для спрощення розрахунків у процесі побудови зображення на екрані, нормалі зазвичай мають одиничну довжину. Очевидно, що нижню частину можна відкинути та отримати досить просту формулу:

E:\ForLoad\Win7Load\prot\f2.jpg

Нормалі завжди рахуються до побудови зображення, щоб зекономити ресурси комп’ютера.[<http://www.alexeyspace.ru/articles/3/>]



Рисунок текстура

**Масив координат текстури**, або так званий uv-масив. Він містить відносні (x, y) координати текстури, що можуть приймати значення від 0 до 1. Вектор (0, 0) відповідає лівому нижньому куту зображення текстури, а (1, 1) верхньому правому. На рисунку зображено три точки з відповідними координатами (x1, y1), (x2, y2), (x3, y3). Заштрихована трикутна область буде «натягнена» на полігон у вершинах якового вказані відповідні uv-координати.

**Масив кольорів**. Для кожної вершини задає колір зазвичай в 24х бітному форматі, хоча формат кольору може змінюватись залежно від системи. Колір вказується для кожної вершини окремо, та буде поступово змінюватись при переході між точками з різними кольорами.

**Масив індексів**. На відміну від усіх вищезгаданих масивів, де кожний елемент відповідає одній точці, даний масив має три елементи для кожного трикутника. Наприклад для побудови квадрату (рис) необхідно 2 трикутники, тобто шість вершин. Однак при цьому можна отримати надлишкові дані, пов’язані з тим, що дві вершини мають однакові координати (B1 = B2, D1 = D2). Інший спосіб передбачає використання 4 вершин, однак з’являється необхідність використання деяких засобів, що дозволять однозначно встановити до яких трикутників належить вершина.



Рисунок квадрат

Кожен елемент масиву індексів це номер вершини в масиві вершин (цифри у кружечках). За кожними трьома елементами в масиві індексів буде побудований один трикутник (індекси 1, 2, 3 для першого і 1, 3, 4 для другого трикутника). Тут існує одна особливість: всі трикутники необхідно описувати за годинниковою стрілкою, якщо дивитись з камери (рис квадрат). Справа в тому, що це дозволяє відображати лише ті полігони, що «дивляться» у камеру. Трикутники описані за стрілками будуть видимі у такому положенні, а якщо описати їх в зворотному порядку то вони будуть видмі лише з іншого боку.

Інша особливість використання індексів полягає в тому, що нормалі вказуються для кожної точки і при використанні однієї точки в декількох полігонах вони обидва матимуть лише одну нормаль. У випадку з квадратом цього не помітно, однак якщо між площинами нормалей є значний кут може виникнути несподіваний ефект освітлення(Рисунок)



Рисунок

На першому рисунку зображено октаедр зі спільними вершинами, на другому для кожного трикутника вказано власні координати вершин, тобто деякі дублюються. У другому випадку нормалі кожного полігону перпендикулярні площині цього полігону, тому добре видно чіткі грані. У першому — вказано одну нормаль і рейдеру доводиться інтерполювати нормалі для кожної точки. Це призводить то ефекту візуального згладжування різких переходів і зазвичай використовується для сфер або поверхонь з плавним переходом.[<https://habrahabr.ru/post/194620/>]

Процедурана генерація

**Рисунок карта висот**

спеціальний рейдер для кольорів

<http://answers.unity3d.com/questions/391561/create-a-mesh-and-color-cubes.html>

Кольори веселки

пристрій лише один з можливих джерел даних (реалізує інтерфейс)