Технічні науки

УДК 004.946

Марков Дмитро Костянтинович

студент

Інституту прикладного системного аналізу

Національного технічного університету України

«Київський політехнічний інститут імені Ігоря Сікорського»

Марков Дмитрий Константинович

студент

Института прикладного системного анализа

Национального технического университета Украины

«Киевский политехнический институт имени Игоря Сикорского»

Markov Dmitriy

Student of the

Institute for Applied System Analysis of the

National Technical University of Ukraine

«Igor Sikorsky Kyiv Polytechnic Institute»

ПОКРАЩЕНЕ ОКЛЮЗИВНЕ ВІДСІЧЕННЯ В СУЧАСНИХ КОМП'ЮТЕРНИХ ІГРАХ

УЛУЧШЕННОЕ ОКЛЮЗИВНОЕ ОТСЕЧЕНИЕ В СОВРЕМЕННЫХ КОМПЬЮТЕРНЫХ ИГРАХ

BETTER OCCLUSION CULLING IN MODERN COMPUTER GAMES

Анотація. *В даній роботі було розглянуто сьогоднішні технології оклюзивного виключення та було запропоновано покращення швидкодії одного з алгоритмів, які використовуються.*

Результатом роботи буде пропозиція покращення існуючого алгоритму BSP та порівняльна характеристика запропонованого покращення відносно базової версії.

Ключові слова: *оклюзивне виключення, рендер, оптимізації рендеру, MOBSP.*

Аннотация. *В данной работе было рассмотрено сегодняшние технологии оклюзивного отсечения и было предложено улучшение быстродействия одного из алгоритмов которые используются.*

Результатом работы будет предложение улучшения существующего алгоритма BSP и сравнительная характеристика предложенного улучшения относительно базовой версии.

Ключевые слова: *оклюзивное отсечение, рендер, оптимизации рендера, MOBSP.*

Summary. *In this paper different modern occlusion culling algorithms were considered and was proposed performance improve for one of the algorithms.*

The results of the paper are proposition about improve of existing algorithm BSP (binary space partitioning) and comparative characteristic of performance gain in improved algorithm.

Key words: *occlusion culling, render, render optimization, MOBSP.*

Постановка проблеми. Рівень якості графічної складової в сучасних комп'ютерних іграх постійно зростає і для того, щоб було можливо відповідати цим стандартам потрібно знаходити можливості для оптимізації.

Аналіз останніх досліджень і публікацій. Дослідження складають праці таких компаній, як Umbra Software [1; 2] та Computer Graphics Laboratory (Zurich) [4].

Формулювання цілей статті (постановка завдання). Аналіз існуючих алгоритмів оклюзивного відсічення та спроба покращення одного з алгоритмів, а саме BSP.

Виклад основного матеріалу. На сьогоднішній день комп'ютерні ігри стали дуже популярними і для того щоб задовольнити потреби нових користувачів ігри стають все більш красивими, реалістичними, живими та складними. Стає більше предметів, деталей, персонажів в будь якій новітній грі. Кожен об'єкт має візуальну складову – меш, набір полігонів, які потрібно відрендерити щоб отримати бажану картинку. Сьогодні меши, які складаються з десятків тисяч полігонів вже ні в кого не викликають подиву. А таких мешів можуть бути сотні, або навіть тисячі і рендерити їх потрібно, як мінімум, 30 разів в секунду, а краще 60 разів. Якщо не звертати на це уваги, то дуже скоро можна виявити, що навіть коли об'єктів не видно, вони все одно витрачають ресурси на своє оновлення, передачу даних на відеокарту та рендер.

Саме тому рендер потребує дуже серйозних оптимізаційних алгоритмів. Багато що в цьому плані вже зроблено виробниками відеокарт і працює в апаратному режимі. Проте оптимізації можна робити набагато раніше, наприклад алгоритм Frustum Culling (відсічення по області видимості) не віддає на рендер об'єкти, які не знаходяться в полі зору камери – це проста і при цьому дуже ефективна оптимізація.

Occlusion Culling це функція, яка відключає рендеринг тих об'єктів, які в дані момент не бачить камера (вони закриті іншими об'єктами). У комп'ютерній 3D графіці це не відбувається автоматично. Найчастіше спочатку рендеряться об'єкти, розташовані далі від камери і вже поверх них рендеряться ближні до камери об'єкти (це називається "overdraw"). Occlusion Culling відрізняється від Frustum Culling. Frustum Culling відключає тільки рендеринг об'єктів, що не потрапляють в область огляду камери, не чіпаючи при цьому приховані за overdraw об'єкти.

Ієрархічний Z-Буферинг та Алгоритм Ієрархічної Видимості

Одним з підходів до оклюзивного відсічення є алгоритм ієрархічної видимості (HV) [Greene93]. Цей алгоритм підтримує модель сцени в октодереві і Z-буфер кадру як піраміду зображення, яку ми називаємо Z-пірамідою. Октодерево дозволяє ієрархічно очистити сховані області сцени, а Z-піраміда забезпечує ієрархічну Z-буферизацію окремих примітивів та обмежує обсяги. Таким чином, Z-піраміди є оклюзійним поданням цього алгоритму.

Будь-який метод може бути використаний для організації примітивів сцени в октодерево, хоча і Greene [Greene93] рекомендує спеціальний алгоритм, який дозволяє уникнути присвоєння малих примітивів великим вузлам. Взагалі, октодерево будується шляхом вкладання всієї сцени в мінімальну коробку, орієнтовану на осі. Решта процедур є рекурсивною за своєю природою і починається перевіркою, чи містить коробка менше порогового числа примітивів. Якщо це так, алгоритм прив'язує примітиви до коробки, а потім завершує рекурсію. В іншому випадку вона розділяє поле на основні осей за допомогою трьох площин, утворюючи таким чином вісім коробок (звідси і назва). Кожна нова коробка тестується і, можливо, знову підрозділяється на 2x2x2 менших ящиків. Цей процес триває, доки кожна коробка містить менше порогового числа примітивів, або до тих пір, поки рекурсія не досягне вказаного найглибшого рівня [Samet89a, Samet89b]. Це проілюстровано у двох вимірах, де структура даних називається квадратним, на мал. 1.

Побудова октодерева займає надто багато часу, щоб зробити його під час виконання, тому цей метод найкраще підходить для статичних моделей.

Після того, як октодерево було створено, кожен кадр виводиться приблизно в прямий порядок, викликаючи процедуру ProcessOctreeNode з кореневим вузлом октодерева. Окружні вузли, що знаходяться за межами

перегляду, зникають. Перший крок визначає, чи обмежується вікно вузла щодо Z-піраміди, використовуючи процедуру, яка буде описана пізніше. У цьому випадку обмежувальним полем вузла є поле в октодереві. Якщо вузол оклюдиться, нам не потрібно більше обробляти вміст цього поля, оскільки його вміст не сприяє остаточному зображенню. В іншому випадку ми відтворюємо примітиви, пов'язані з вузлом, у Z-піраміду, а потім обробляємо кожен з дітей вузла (якщо він має будь-який), використовуючи цю ж рекурсивну процедуру. Коли завершується рекурсія, всі видимі примітиви були вписані в Z-піраміду, і було створено стандартний образ Z-буфера сцени.

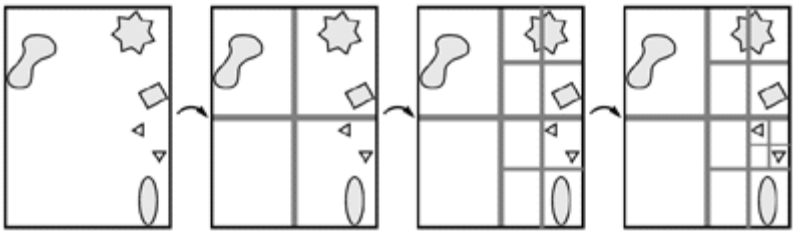

Рис. 1. Побудова квад-дерева (що є двовимірною версією октави). Будівництво починається зліва, вклавши всі об'єкти у обмежувальну коробку. Потім коробки рекурсивно поділяють на чотири коробки з однаковим розміром

Алгоритм HV дуже ефективно виконує оклюзію, оскільки він перетинає тільки видимі вузли з октодерева і їхніх дітей, і він відображає лише примітиви у видимих вузлах. Це може заощадити більшу частину роботи на сценах, які щільно закриті. Наприклад, у сцені, може бути такая ситуація, що більше 99% екранних багатокутників знаходяться всередині замкнутих вузьких вузлів, які, таким чином, вибираються пірамідою Z [Greene95].

```
1: ProcessOctreeNode(OctreeNode N)
2: if(isOccluded(NBV, ZP)) then return;
3: for each primitive p in N
4: tileInto(p, ZP)
5: end
6: for each child node C in N in front-to-back order
7: ProcessOctreeNode(C)
8: end
```

Тепер ми опишемо, як підтримується Z-піраміда і як вона використовується для прискорення оклюзії. Найкращий вид Z-піраміди - це просто стандартний Z-буфер. На всіх інших рівнях кожне значення z є найдальшим z у відповідному вікні 2x2 сусіднього більш точного рівня. Тому кожне значення z представляє найдальший z для квадратної області екрана. Щоб підтримувати Z-піраміду, коли Z-значення перезаписується в Z-буфері, воно поширюється через грубіші рівні Z-піраміди. Це робиться рекурсивно, доки не буде досягнута верхня частина піраміди зображення, де залишається лише одне значення z (це показано на рис. 2).

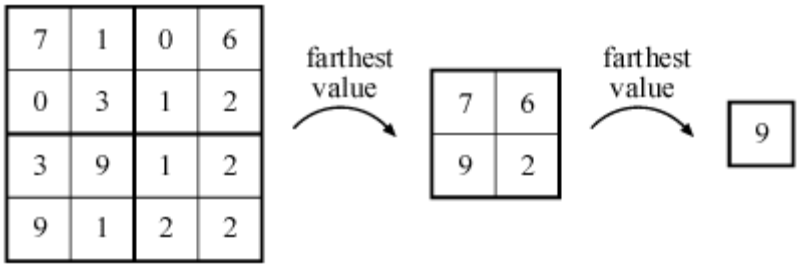

Рис. 2. Зліва відображається фрагмент Z-буфера розміром 4x4. Числові значення - це фактичні z-значення. Це зменшено до області 2x2, де кожне значення є найдальшим (найбільшим) з чотирьох областей 2x2 зліва

Ієрархічне виключення за допомогою оклюзивних дерев (MOBSP + SVBSP)

Опис алгоритму

Даний алгоритм, вирішує проблему консервативної видимості з точки (точки зору). Він ідентифікує суперсет об'єктів, видимих з точки зору. Для складних сцен, де більшість об'єктів не видна із точки зору, суперсет буде тільки частиною сцени. Точна видимість вирішується просто за допомогою рендерингу видимих об'єктів з використанням алгоритму Z-буфера. Припустимо, що ми можемо визначити видимість області від точки зору. Ця видимість означає одне із: повністю видимий, частково видимий, невидимий. Ми можемо застосувати тест видимості на всі

обмежуючі фігури об'єктів на сцені. Проте, для складної сцени тестування всіх об'єктів потребувало б дуже багато часу.

Ми можемо використовувати просторову когерентність видимості шляхом угруповання об'єктів на близькій відстані разом [3]. Застосовуючи цей крок рекурсивно, ми можемо побудувати просторову ієрархію, зберігаючи посилання на об'єкти в його листових вузлах. Кожен вузол

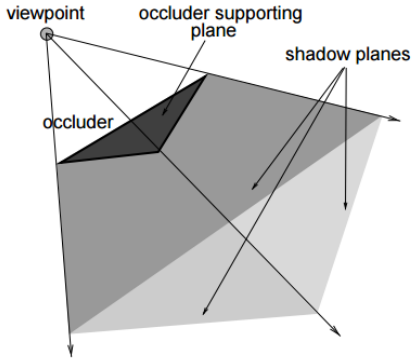

Рис. 3. Оклюдер та його тінь [15]

ієрархії відповідає певному діапазону. Починаючи з кореневого вузла ієрархії, видимість кожного вузла може бути визначена наступним чином: якщо знайдений вузол повністю видно, всі його нащадки повністю видимі. Точно так же, якщо знайдений вузол невидимий, всі його діти є невидимими. Нащадки вузлів які визначено частково видимими повинні бути додатково перевірені. Коли всі листові вузли, що визначені як видимі чи частково видимі, зібрані, їх можна візуалізовувати з використанням низькорівневих оклюдерів (апаратне забезпечення з Z-буфером).

Залишається показати, як визначити видимість регіону з точки зору. Часто буває так, що велика частина оклюзії відбувається через декількох великих об'єктів (оклюдерів), близьких до точки зору. У даній роботі оклюдером може бути тільки опуклий багатокутник. Припустимо, ми можемо виділити кілька таких оклюдером для кожної точки. Для кожного полігону оклюзивне фігура може бути визначена. Це перетин (e + 1) в сумі половини просторів, де e число ребер багатокутника. Підпростори утворені

площинами, що проходять через точки зору і конкретного краю і опорною площині багатокутника. Об'єднуються ці фігури в одну структуру - оклюзивне дерево, тобто варіант дерева бінарного розбиття простору (BSP дерева). Видимість замкнутої багатогранної області може бути визначена шляхом комбінування станів видимості з його граней. Регіони нашої просторової ієрархії вирівняний по осях коробки (паралелепіпеди), які представляють собою замкнуті багатогранники з шістьма опуклими гранями. В рамках цього алгоритму також представлено модифіковане оклюзивне дерево (MOBSP). При цьому структура даних видимості області може бути створена без тестування видимості її кордонів (граней). Єдина операція, що бере участь у перевірці видимості, є визначення положення в області щодо площині. Хоча цей метод може ідентифікувати невидиму область як частково видиму (по відношенню до обраних оклюдерів), було спостережено його хорошу продуктивність на практиці.

В звичайному BSP база оклюдерів використовує препроцесіювання для створення. Простір поділяється на безліч непересічних областей. Усередині кожної клітини певну кількість багатокутних оклюдерів визначаються і зберігаються. Ми не намагаємося побудувати таку базу даних оклюдерів. Замість цього визначаються потенційні багатокутники-оклюдери. В даній реалізації вони визначені в своїх інтересах знання модельної структури. Препроцесіювання і виключення за видимістю були застосовані, як правило, на моделях архітектурних інтер'єрів. Типова модель складається зі стін, стель, підлог і деталізованих об'єктів. Всі полігони, що належать до деталізованих об'єктів (квіти, стільці, ...) вважаються неоклюдуючими. Всі інші багатокутники позначені як потенційні оклюдери (припускаючи стіни, стелі й підлоги). Ці потенційні оклюдери використовуються в алгоритмі динамічного вибору оклюдерів, а потім для визначення видимості [7].

Як вже згадувалось, алгоритм ієрархічної видимості передбачає, що просторова ієрархія будується над усіма об'єктами моделі. У разі статичних сцен це може бути зроблено в попередній обробці. Існує важлива вимога, що пред'являється до ієрархії. Області, що відповідають нащадкам будь-якого вузла ієрархії повинна бути повністю міститься в області, що відповідає цьому вузлу. В іншому випадку, ніяких припущень про видимість нащадків вузла не може бути зроблено на основі знань про видимість їх батьків. В даному варіанті було використано октодерево (BSPtree). Використовується вирівняне по осях BSP дерево (іноді називають KD-дерево), через свою високої гнучкості і простоти побудови і обходу. Цей вибір має на увазі, що області, відповідні вузлам ієрархії є паралелепіпеди. Природно, що BSP дерево відповідає критерію, що вже згадувався вище. Найважливішим кроком при будівництві дерева BSP є вибір розщеплення площини. Ця площина поділяє поточний вузол в двох нащадків. Об'єкти будуть розподілені в нащадків відповідно до їх становищем з розщепленням площині. Спочатку кореневої вузол дерева BSP відповідає обмежувальної рамки моделі. Застосовуючи алгоритм рекурсивно, будується дерево. Рекурсія завершується, коли кількість об'єктів в поточному вузлі потрапляє під задану кількість або максимальна задана глибина ієрархії досягається. У деяких випадках об'єкт лежить по обидва боки від площини (тобто в позитивному і негативному півпросторі, індукованих площиною). Такі об'єкти повинні бути "дубльовані" в обох нових вузлів. Потрібно, щоб об'єктів дубльованих в листових вузлах дерева, було мінімум, зберігаючи при цьому добре збалансоване дерево. Для досягнення цієї цілі наступна стратегія вибору розщеплення площині був використаний: для поточного вузла визначається вісь з найбільшим ступенем паралелепіпеда, відповідного вузла [1]. Шукається площина розколу, перпендикулярна до обраної осі. Визначено межі об'єкта обмежуючих прямокутників, розташованих на певній відстані від

просторової медіани паралелепіпеда вузла. Оцінюється ряд об'єктів, розщеплених кожною граничною площиною. Бінарна структура дерева може бути легко використана для імітації нерегулярних квадродерев і октодерев в рамках ієрархічного алгоритму видимості [14].

Реалізація алгоритму

Мета динамічного вибору оклюдеру є отримання певного числа оклюдерів, враховуючи точки зору і напрямок погляду. Алгоритм використовує область кута вимірювання для оцінки якості оклюдером. Характеристика оклюдера виражається як:

$$M = \frac{-A(\vec{N} * \vec{V})}{\|\vec{D}\|^2}$$

де A є область оклюдером, $\vec{N}$ позначає нормаль оклюдеру, $\vec{V}$ напрямок зору і $\vec{D}$ відповідає до вектору з точки зору до центру оклюдеру (при цьому вектори нормалі та напрямку зору нормовані) [15].

```
Algorithm FilterDown(Node, Polygon, Viewpoint)
begin
     if Node is leaf then
          if Node is out-leaf then
               replace Node by OcclusionVolume(Polygon, Viewpoint)
          else
               do nothing
     else
          case Split(Polygon, Node.Plane, Back, Front) of
          FRONT : (* pass the polygon to the front subtree *)
          FilterDown(Node.FrontChild, Polygon, Viewpoint);
          BACK : (* pass the polygon to the back subtree *)
          FilterDown(Node.BackChild,Polygon, Viewpoint);
          SPLIT : (* pass fragments to apropriate subtrees *)
          FilterDown(Node.FrontChild, Front, Viewpoint)
          FilterDown(Node.BackChild, Back, Viewpoint);
     end
end
```

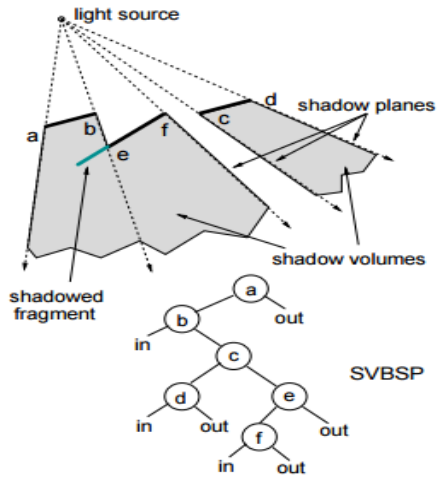

Рис. 4. Візуалізація роботи алгоритму [15]

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі. Після ряду тестів було визначено що покращення існуючого алгоритму ускладнює його роботу та реалізацію в коді. Проте виграш помітний на більшості типів сцен, як і було прогнозовано в теорії. Оптимізація дозволяє отримати більш насичену об'єктами та моделями сцену залишаючись на тому самому фрейм-рейті. Приріст швидкодії залежить від вдалості вибору оклюдерів та індивідуальних параметрів сцени.

Література

1. Next generation occlusion culling. – Режим доступу: http://www.gamasutra.com/view/feature/164660/sponsored_feature_next_generation_.php?print=1. – Дата доступу: 04.06.16
2. GDC Vault. – Режим доступу: http://gdcvault.com/free/gdc-15. – Дата доступу: 04.06.16
3. Краткий курс компьютерной графики, аддендум: GLSL. – Режим доступу: https://habrahabr.ru/post/253791/. – Дата доступу: 04.06.16

4. GPU-Based Ray-Casting of Quadratic Surfaces. – Режим доступу: http://reality.cs.ucl.ac.uk/projects/quadrics/pbg06.pdf. – Дата доступу: 04.06.16
5. OpenGL 44 Pipeline Map. – Режим доступу: http://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGL44PipelineMap.pdf – Дата доступу: 04.06.16
6. Dynamic Scene Occlusion Culling using Regular Grids. – Режим доступу: http://www.dca.fee.unicamp.br/projects/mtk/batageloM/. – Дата доступу: 04.06.16
7. Occlusion Culling Algorithms. – Режим доступу: http://www.gamasutra.com/view/feature/131801/occlusion_culling_algorithms.php?page=2. – Дата доступу: 04.06.16
8. Occlusion Culling Algorithms. – Режим доступу: http://www.gamasutra.com/view/feature/3394/occlusion_culling_algorithms.php?print=1. – Дата доступу: 04.06.16
9. Software Occlusion Culling. – Режим доступу: https://software.intel.com/en-us/articles/software-occlusion-culling. – Дата доступу: 04.06.16
10. Efficient Occlusion Culling. – Режим доступу: http://http.developer.nvidia.com/GPUGems/gpugems_ch29.html. – Дата доступу: 04.06.16
11. Удаление невидимых поверхностей. Алгоритм, использующий Z-буфер. – Режим доступу: http://opita.net/node/58. – Дата доступу: 04.06.16
12. Удаление невидимых поверхностей: z-буфер. – Режим доступу: https://habrahabr.ru/post/248179/. – Дата доступу: 04.06.16

References

1. Next generation occlusion culling. – Access: http://www.gamasutra.com/view/feature/164660/sponsored_feature_next_generation_.php?print=1. – Date : 04.06.16
2. GDC Vault. – Access: http://gdcvault.com/free/gdc-15. – Date : 04.06.16
3. Short course of computer graphics: GLSL. – Access: https://habrahabr.ru/post/253791/. – Date : 04.06.16
4. GPU-Based Ray-Casting of Quadratic Surfaces. – Access: http://reality.cs.ucl.ac.uk/projects/quadrics/pbg06.pdf. – Date : 04.06.16
5. OpenGL 44 Pipeline Map. – Access: http://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGL44PipelineMap.pdf – Date : 04.06.16
6. Dynamic Scene Occlusion Culling using Regular Grids. – Access: http://www.dca.fee.unicamp.br/projects/mtk/batageloM/. – Date: 04.06.16
7. Occlusion Culling Algorithms. – Access: http://www.gamasutra.com/view/feature/131801/occlusion_culling_algorithms.php?page=2. – Date : 04.06.16
8. Occlusion Culling Algorithms. – Access: http://www.gamasutra.com/view/feature/3394/occlusion_culling_algorithms.php?print=1. – Date : 04.06.16
9. Software Occlusion Culling. – Access: https://software.intel.com/en-us/articles/software-occlusion-culling. – Date : 04.06.16
10. Efficient Occlusion Culling. – Access: http://http.developer.nvidia.com/GPUGems/gpugems_ch29.html. – Date: 04.06.16
11. Invisible surface removal algorithm – Access: http://opita.net/node/58. – Date : 04.06.16
12. Invisible surface removal algorithm: z-buffer. – Access: https://habrahabr.ru/post/248179/. – Date : 04.06.16