Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт (государственный университет)»
Физтех-школа прикладной математики и информатики
Центр обучения проектированию и разработке игр

Направление подготовки: 09.04.01 Информатика и вычислительная техника

Направленность (профиль) подготовки: Анализ данных и разработка информационных систем

Архитектура рендеринга реального времени через вычислительный граф

(магистерская диссертация)

Студент:

Санду Роман Александрович

_____

(подпись студента)

Научный руководитель:

Щербаков Александр Станиславович

_____

(подпись научного руководителя)

Москва 2023

Аннотация

Данная работа посвящена одному из подходов к построению архитектуры приложений реального времени, называемого неформально ”фреймграфом” или ”рендерграфом”. Подход основывается на использовании вычислительного графа как представления процесса вычисления итоговой картинки одного кадра приложения.

Содержание

1. Введение

1.1. Аллокация ресурсов

В процессе вычисления картинки одного кадра любое нетривиальное приложение использует *транзиентные ресурсы* – промежуточные хранилища данных, содержимое которых не требуется после окончания вычисления кадра. Основная отличительная черта рассматриваемого подхода заключается в известности всей информации о транзиентных ресурсах заранее, что позволяет управлять ими более эффективно. Более того, наш подход позволяет добиться в определённом смысле оптимальной работы с такими ресурсами, как будет видно дальше. Однако обязательным пререквезитом для эффективной аллокации ресурсов является использование современного графического API, предоставляющего возможность ручного управления видеопамятью. До появления подобных API большая часть приложений использовало один из следующих наивных подходов к управлению ресурсами.

Самый простым подходом является выделение и освобождение транзиентных ресурсов по ходу их нужды при помощи соответствующих вызовов графического API. Этот подход сильно похож к управлению памятью объектов в системных языках программирования: драйвер операционной системы содержит аллокатор, на который пользователь перекладывает обязанность управления памятью и другими ресурсами GPU, аналогично куче в языке C. Системный аллокатор переиспользует освободившуюся память, тем самым достигая низкого её потребления. Однако такой подход не масштабируется на более сложные приложения. (фрагментация, отложенное удаление, рефкаунтинг, етц)

Альтернативным подходом служит отказ от переиспользования памяти. Все транзиентные ресурсы создаются заранее и не удаляются в ходе работы приложения. ...

Наконец, наиболее практичным подходом является пулинг ресурсов. ...

2. Обзор существующих работ

2.1. Имплементации

Frostbite

Первыми идею организации архитектуры рендеринга в приложениях реального времени через вычислительные графы предложили разработчики движка Frostbite в 2017 году[1]. *Кадровый граф* позволил им сделать ядро модуля рендеринга расширяемым, упростил работу с асинхронным вычислениями общего назначения на GPU, автоматизировал работу со специализированными видами оперативной видеопамяти на игровых консолях, а также сэкономил ”тонны” обычной видеопамяти. В силу проприетарности движка не известно, насколько широкий класс сценариев использования ресурсов она поддерживает. В качестве схемы аллокации ресурсов же был взят обычный онлайн-аллокатор, располагающий в заранее выделенном крупном участке памяти ресурсы по мере необходимости. Автоматическая расстановка барьеров на 2017 год не поддерживалась.

Halcyon

Далее, в 2019 году, компания EA представила[2] новый экспериментальный движок Halcyon, обобщающий идею кадрового графа до *графа рендеринга*.

Unity

документация[3] закрытая, но вроде хорошая

Unreal Engine

документация[4]

Anvil

Ubisoft выступление[5] есть алиасинг, есть автобарьеры (сплит), умеет в несколько очередей сабмита

Granite

блог[6]

Прочие

Неинтересные: https://github.com/azhirnov/FrameGraph – нет алиасинга, очень много ООП, намертво привязан к вулкану, вершины не реордерятся, содержимое вершин – фиксированные таски, а не произвольный код, нет хистори ресурсов, есть барьеры, ВРОДЕ БЫ нет алиасинга https://github.com/skaarj1989/FrameGraph – нет алиасинга, нет хистори ресурсов, нет барьеров, кросс-АПИ, прикольный интерфейс на C++, видимо заброшен https://github.com/Raikiri/Leg – ОТЕЧЕСТВЕННОЕ!!!

2.2. Аллокация ресурсов

Задача поиска расписания аллокации ресурсов в графе кадра в своей простейшей формулировке является классической сильно NP-сложной[7] задачей *динамической аллокации памяти* (dynamic storage allocation, DSA[8, с. 226]). У этой задачи существует две интерпретации, он-лайн и офф-лайн. Первая подразумевает обработку разнесённых во времени запросов на аллокацию и деаллокацию ресурсов, иначе говоря, решения об адресах ресурсов в памяти необходимо принимать в порядке времён появления ресурсов. Этот частный случай часто встречается в операционных системах и рантаймах языков программирования. Вторая же интерпретация подразумевает наличие заранее известных времён жизни всех ресурсов. В рамках данной работы нас интересует именно офф-лайн интерпретация, поэтому, в отсутствие уточнения, под задачей о динамической аллокации памяти мы будем подразумевать именно её.

Одним из первых полиномиальных алгоритмов предложенных для решения задачи DSA является алгоритм First-Fit[9], работающий, как было вскоре доказано Кирстедом, с константной ошибкой не более чем в 80 раз[10]. Тремя годами позже Кирстед представил алгоритм с ошибкой не более чем в 6 раз[11]. Эти и другие ранние работы объединяет общий подход сведения DSA к частному случаю с единичным размером всех ресурсов, эквивалентному покраске интервального графа, и последующим применением он-лайн алгоритма покраски. Через несколько лет Йордан Гергов, отказавшись от сведения к интервальным графам, смог понизить верхнюю оценку минимальной возможной ошибки до 5[12], а в последствии и до 3[13]. Наконец, наилучший на данный момент результат был получен исследователями из AT&T Labs совместно с коллегой из Ecole Polytechnique[14]: полиномиальный алгоритм, для любого заранее выбранного ε дающий $(2 + \varepsilon)$-приблизительное решение DSA. Более

того, для некоторых частных случаев авторы предоставляют приближённую схему полиномиального времени (то есть $(1+\varepsilon)$-приближение). Из них в рамках графа кадра особо интересна схема для случая ресурсов, размер которых ограничен сверху константой h_{max}. Однако практичность представленных алгоритмов в рамках приложений реального времени является открытым вопросом в силу их высокой сложности (TODO: оценить асимптотику по мастер-теореме).

Похожая задача, как бы это не было удивительно, возникает в области оперирования морских контейнерных терминалов. С ростом сложности и нагруженности глобальных транспортных цепочек, прикладные задачи оперирования верфей стали слишком сложны для интуитивного их решения. В связи с этим за последние несколько десятилетий было сформулировано и в той или иной степени решено множество вариаций *задачи об аллокации верфи*, покрывающих широкий спектр прикладных задач. Так как расписания прибытия кораблей обычно известно портам заранее, офф-лайн задача динамической аллокации памяти является частным случаем одной из формулировок этой задачи, а именно вариации $cont|dyn|fix|max(res)$ по классификации обзорной статьи Бирвирта и Мизла[15]. Именно из-за этого задача об аллокации верфи представляет интерес в рамках данной работы.

Одним из первых интересующую нас формулировку задачи об аллокации верфи рассмотрел в своей статье Эндрю Лим[16]. Ресурсы, имеющие фиксированные и известные размер и времена аллокации и деаллокации, могут быть рассмотрены как корабли с соответствующей длинной, временем прибытия и временем отплытия, а тип используемой видеопамяти как секция верфи. Задача нахождения минимальной длинны всех секций верфи и точек прибытия всех кораблей аналогична нахождению минимального необходимого объёма памяти и локаций всех ресурсов в этой памяти. Однако, в отличии от рассматриваемой Лимом задачи, ресурсы не накладывают требований на отступ между друг другом и началом или концом верфи, зато требуют определённого выравнивания их начала в памяти. Впрочем, последние условие достаточно легко сводится к первому.

Однако в данной работе рассматривается более общая формулировка задачи об аллокации ресурсов, насколько известно авторам, не рассматривавшаяся ранее в литературе.

Список литературы

1. *O'Donnell Y.* FrameGraph: Extensible Rendering Architecture in Frostbite. — 2017. — URL: https://www.gdcvault.com/play/1024612 ; Game Developers Conference.
2. *Wihlidal G.* Halcyon: Rapid innovation using modern graphics. — 2019. — URL: https://www.youtube.com/watch?v=da_6dsWz8yg ; Reboot Develop.
3. *Technologies U.* Unity render graph system. — URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.core%4014.0/manual/render-graph-system.html.
4. *Games E.* Unreal Engine rebder dependency graph. — URL: https://docs.unrealengine.com/5.0/en-US/render-dependency-graph-in-unreal-engine/.
5. *Gruen H.* DirectX™ 12 Case Studies. — 2017. — URL: https://www.gdcvault.com/play/1024343 ; Game Developers Conference.
6. *Arntzen H.-K.* Render graphs and Vulkan — a deep dive. — 2017. — URL: https://themaister.net/blog/2017/08/15/render-graphs-and-vulkan-a-deep-dive/.
7. *Stockmeyer I. J.* — 1976. — личная переписка.
8. *Garey M. R.*, *Johnson D. S.* Computers and Intractability; A Guide to the Theory of NP-Completeness. — USA : W. H. Freeman & Co., 1990. — ISBN 0716710455.
9. *Chrobak M.*, *Ślusarek M.* On some packing problem related to dynamic storage allocation // RAIRO - Theoretical Informatics and Applications. — 1988. — Vol. 22, no. 4. — P. 487–499. — ISSN 0988-3754, 1290-385X. — DOI: 10.1051/ita/1988220404871. — URL: http://www.rairo-ita.org/10.1051/ita/1988220404871 (visited on 10/23/2022).
10. *Kierstead H. A.* The Linearity of First-Fit Coloring of Interval Graphs // SIAM Journal on Discrete Mathematics. — 1988. — Nov. — Vol. 1, no. 4. — P. 526–530. — ISSN 0895-4801, 1095-7146. — DOI: 10.1137/0401048. — URL: http://epubs.siam.org/doi/10.1137/0401048 (visited on 10/23/2022).
11. *Kierstead H. A.* A polynomial time approximation algorithm for dynamic storage allocation // Discrete Mathematics. — 1991. — Т. 88, № 2. — С. 231—237. — Publisher: Elsevier.
12. *Gergov J.* Approximation algorithms for dynamic storage allocation // European Symposium on Algorithms. — Springer, 1996. — С. 52—61.

13. *Gergov J.* Algorithms for compile-time memory optimization // Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms. — 1999. — С. 907—908.
14. OPT versus LOAD in dynamic storage allocation / A. L. Buchsbaum [и др.] // Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. — 2003. — С. 556—564.
15. *Bierwirth C.*, *Meisel F.* A survey of berth allocation and quay crane scheduling problems in container terminals // European Journal of Operational Research. — 2010. — Т. 202, № 3. — С. 615—627. — ISSN 0377-2217. — DOI: https://doi.org/10.1016/j.ejor.2009.05.031. — URL: https://www.sciencedirect.com/science/article/pii/S0377221709003579.
16. *Lim A.* The berth planning problem // Operations Research Letters. — 1998. — Т. 22, № 2. — С. 105—110. — ISSN 0167-6377. — DOI: https://doi.org/10.1016/S0167-6377(98)00010-8. — URL: https://www.sciencedirect.com/science/article/pii/S0167637798000108.