Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Николаев Кирилл Викторович

Методы оптимизации мобильных интерфейсов

Научный руководитель: Филиппов Василий Борисович,
кандидат физико-математических наук,
руководитель отдела мобильной дистрибуции, ООО «Яндекс».

Санкт-Петербург
2012

Оглавление

1. Введение....................2
   1.1. Актуальность мобильных интерфейсов....................2
   1.2. Технологии....................3
   1.3. Суть работы....................4
   1.4. Структура работы....................5
2. Обзор методов....................6
   2.1. Возможные направления оптимизаций....................6
      2.1.1. Оптимизация текстур....................6
      2.1.2. Использование текстурных атласов....................7
      2.1.3. Оптимизация пиксельных и вершинных шейдеров....................7
      2.1.4. Эффективная эксплуатация кеша GPU....................7
      2.1.5. Оптимизация смены состояний....................7
      2.1.6. Группировка геометрий....................7
   2.2. Выбранный метод....................8
3. Постановка задачи....................9
   3.1. Общая идея....................9
      3.1.1. Наивный метод....................10
      3.1.2. Сохранение порядка....................11
   3.2. Формальная постановка задачи....................11
4. Анализ задачи....................13
   4.1. Основные определения....................13
   4.2. Задача о кратчайшей общей надпоследовательности....................15
   4.3. Случай неповторяющихся символов....................16
   4.4. NP-полнота задачи упорядочения....................17
   4.5. Связанные задачи....................19
5. Предлагаемый алгоритм....................22
   5.1. Требования....................22
   5.2. Специальные случаи....................22
   5.3. Псевдо-код....................24
   5.4. Анализ....................25
6. Тестирование....................27
   6.1. Выигрыши, обусловленные объединением геометрий....................27
   6.2. Выигрыши, обусловленные группировкой по материалам....................27
7. Заключение....................29
8. Источники....................30

1. Введение

1.1. Актуальность мобильных интерфейсов

Рис. 1. Пример современного мобильного приложения: SPB Shell 3D ([SHELL])

Буквально за несколько прошедших лет положение мобильных телефонов и смартфонов в жизни людей совершенно изменилось. Еще недавно практически единственной широко используемой функцией телефонов, была непосредственно голосовая и текстовая связь. Смартфоны — телефоны с возможностью расширения функциональности устройства путем установки дополнительных приложений пользователем — использовались лишь энтузиастами и бизнес-пользователями. Это было обусловлено взаимосвязанными техническими и маркетинговыми факторами. К техническим факторам можно отнести такие, как низкое быстродействие, низкое разрешение экрана, неудобные методы взаимодействия с пользователем (резистивные сенсорные экраны, требующие использования стилуса). Следствием этого являлась невозможность

создания эффективного пользовательского интерфейса, разработка жизнеспособных сценариев использования, ориентированных на широкие массы пользователей.

Однако, естественное развитие технологии позволило компании Apple выпустить на рынок новую категорию устройства — смартфон, ориентированный на широкий круг пользователей. Одним из главных факторов, повлиявшим на интерес рынка к новому устройству, являлся продуманный и быстрый интерфейс. Именно продуманные сценарии работы и отзывчивость пользовательского интерфейса Apple и сделал главным приоритетом разработки iPhone. Этот пример положил начало бурному росту рынка смартфонов, который привел к тому, что продажи смартфонов в последние годы превышают продажи персональных компьютеров [IPHONE]. Пользователи мобильных устройств стали основными потребителями электронного контента. Поэтому важной темой изысканий сегодняшнего дня является поиск способов эффективного взаимодействия с мобильными пользователями, улучшение и оптимизация интерфейсов. В соответствии с требованиями сегодняшнего дня настоящая работа осветит некоторые вопросы, связанные с оптимизацией производительности мобильных интерфейсов.

1.2. Технологии

Именно Apple при разработке iPhone начала применять технологии, ранее используемые для игр и сложной визуализации данных, для построения пользовательского интерфейса мобильных устройств. *OpenGL* — это универсальный кросс-платформенный программный интерфейс для доступа к функциям графического адаптера. OpenGL оперирует абстракциями достаточно низкого уровня по сравнению абстракциями библиотек пользовательского интерфейса и менеджера окон. В современных мобильных устройствах на базе Android и iOS любое отображение информации на экран происходит с использованием OpenGL. Оконный менеджер использует его для смешивания слоев окон и элементов управления, а также для анимации эффектов переходов между

экранами.

Для прикладного программиста эти ОС обычно предоставляют набор примитивов более высокого уровня, состоящий из готовых элементов управления, заготовленных эффектов, фиксированных анимаций. Однако, эти абстракции не бесплатны с точки зрения быстродействия и возможной выразительности. Поэтому для построения действительно богатого и быстрого интерфейса приходится использовать низкоуровневые средства: OpenGL и различные графические платформы, построенные вокруг его концепций.

1.3. Суть работы

В работе мы будем рассматривать именно такой *графический движок* (*graphics engine*), разработанный в компании SPB Software (ООО «Яндекс») для использования в приложениях для ОС Android. Рассматриваемая система следует традициям разработки трехмерных движков [GRAPH] и предоставляет абстракцию *графа сцены* (*scene graph*) вместе с методами его манипуляции и отрисовки. Он является промежуточным звеном между *библиотекой двумерного интерфейса* (*UI Engine*), прикладным кодом и средствами ОС работы с графикой (OpenGL, и, возможно, другие носители).

Разделение архитектуры на эти слои оставляет поле для изменения схемы взаимодействия с ОС и графическим адаптером и сохраняет независимость клиентского кода от деталей реализации. Это тем более важно, если учесть, что разные модели устройств поддерживают разные наборы расширений OpenGL и имеют уникальные тонкости работы *графического процессора* (*GPU*).

Целью работы является реализация методов повышения производительности интерфейса на уровне графического движка. При этом важно обеспечить прозрачность производимых оптимизаций с точки зрения прикладного программиста. Это обусловлено тем, что эффективная работа с OpenGL является непростой задачей, и стратегии этой работы желательно сосредоточить в одном месте.

В этом и заключается новизна работы в силу того, что в большинстве по-

пулярных движков такого рода оптимизации, предлагаемые в работе, не производятся ([IRRLICHT], [LINDERDAUM]) или являются явными ([COCOS], [OGRE]).

1.4. Структура работы

Для достижения поставленных целей мы вначале рассмотрим возможные направления оптимизаций. Некоторые из них окажутся неприменимыми в рассматриваемом контексте или могут заведомо приводить к незначительным результатам.

Далее в работе будет описан метод, который привел к заметному повышению производительности приложения. Этот метод основан на снижении нагрузки на *центральный процессор* (*CPU*) путем объединения нескольких OpenGL команд в *пакеты* (*batching*) и минимизации числа изменений состояния.

Эффективность выбранного метода подтверждается измерениями производительности как для специально изготовленного тестового примера, так и в практических условиях.

2. Обзор методов

2.1. Возможные направления оптимизаций

В работе [TRAPP] предлагается краткий обзор аспектов, которые могут негативно влиять на производительность OpenGL, традиционных методов их устранения вместе с методиками оценки их эффективности.

Основными из них являются:

- уменьшение числа изменений состояния GPU;
- минимизация числа *пакетов вершин* (*batches*);
- эффективное использование кеша GPU;
- удаление объектов, закрытых от обзора;
- уменьшение общего количества информации, отправляемых графическому адаптеру;
- перенос преобразований, осуществляемых для каждого пикселя, в число тех, что выполняются для каждой вершины, а также снижение числа последних;
- оптимизация текстур.

Остановимся на некоторых из предлагаемых методов.

2.1.1. Оптимизация текстур

Включает в себя выбор адекватной глубины цвета и формата данных для снижения общего объема информации и, следовательно, увеличение эффективности кеша. Однако, в рассматриваемом случае этот метод приводит к заметной деградации внешнего вида приложения. К тому же, нет универсального формата сжатых текстур, которые будут приводить к выигрышам на широком спектре устройств.

2.1.2. Использование текстурных атласов

Склеивание нескольких текстур в одну большую для экономии времени загрузки на загрузку и замену текстур при рисовании приводит к отличным результатам [ATLAS]. В рамках данной работы мы еще будем упоминать эту оптимизацию, так как для выбранного нами метода существенно важно, чтобы максимальное число объектов использовали одно и то же графическое состояние.

2.1.3. Оптимизация пиксельных и вершинных шейдеров

Перенос вычислений из пиксельного в вершинный шейдер приводит к значительному повышению производительности, так как количество обрабатываемых пикселей обычно гораздо больше, чем вершин в геометрии. Эта оптимизация проводится один раз при разработке шейдера, поэтому мы не будем затрагивать ее в работе. Методики, позволяющие определить, какую работу можно перенести, описаны, например в [TRAPP].

2.1.4. Эффективная эксплуатация кеша GPU

Семейство оптимизаций, основанных на изменении порядка использования вершин учитывая требования локальности описаны в [ORDER] и [CULL].

2.1.5. Оптимизация смены состояний

Этот метод основан на предположении о том, что изменение состояния GPU, в частности установка активной текстуры — дорогая операция. Идея состоит в том, чтобы объединить объекты, использующие общее состояние, в группы, а затем производить отрисовку групп последовательно. При этом изменение состояния можно производить перед обработкой группы в целом, а не перед отрисовкой каждого объекта. Таким образом, время, затраченное на смену состояния, становится пропорциональным количеству используемых текстур, а не объектов.

2.1.6. Группировка геометрий

Время отправки блока вершин содержит не только затраты для данных

каждой вершины, но и постоянный накладной расход на подготовку самого пакета для отправки видеоадаптеру [BATCH]. Более того, для многих устройств верно, что время отправки пакета слабо зависит от числа вершин в пакете. Это естественным образом приводит нас к необходимости объединять максимальное число вершин в один общий блок и отправлять его графическому адаптеру одним пакетом.

2.2. Выбранный метод

Тестирование показало, что последние два описанные метода (в совокупности с объединением текстур) приводят к наибольшим выигрышам в практических приложениях. Кроме того, для этих вариантов оптимизации возможно проведение преобразований на уровне движка прозрачно для клиентского кода.

3. Постановка задачи

3.1. Общая идея

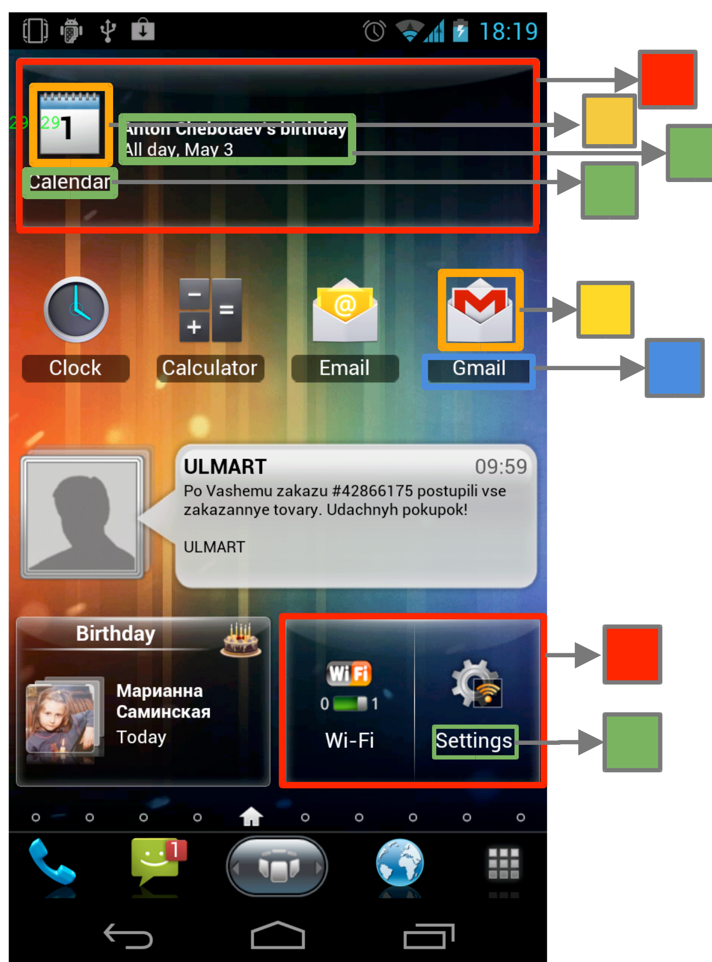

Рис. 2. Примеры отрисовываемых элементов в приложении [SHELL]

Экран мобильного приложения состоит из множества объектов преимущественно простой формы (рис. 2). Однако, в общем случае каждый объект обладает дополнительной информацией: материал (текстура, цвет), шейдер, которым он должен отрисоваться, требуемые режимы отрисовки OpenGL (blending, scissor, и др.). Набор свойств объекта будем называть *состоянием*. Имеется в виду состояние в котором должен быть графический адаптер при отрисовке дан-

ного объекта.

Если производить отрисовку в произвольном порядке (в порядке добавления вершин в граф сцены), то каждый объект потребует дополнительного времени для установки требуемого состояния, выбора текстуры и загрузки вершин, его составляющих. Чтобы минимизировать накладные расходы, сгруппируем объекты так, чтобы последовательные элементы использовали одну и ту же конфигурацию. Таким образом, установка состояния потребуется только один раз на группу. В свою очередь, геометрии объектов, оказавшихся в одной группе, можно объединить.

3.1.1.Наивный метод

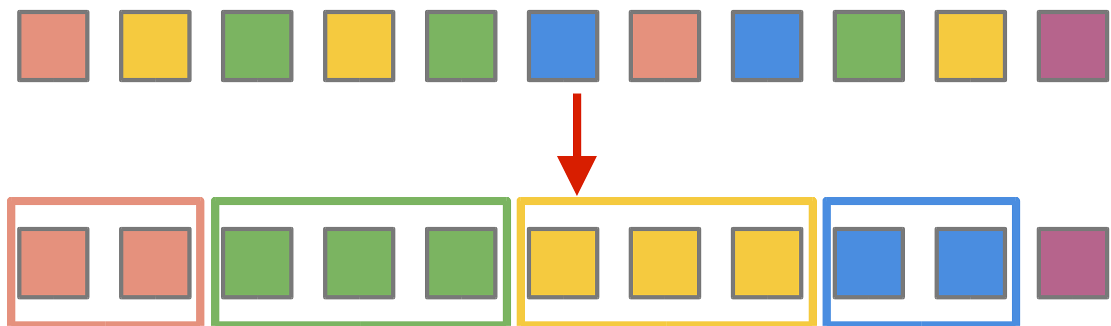

Рис. 3. Пример сортировки наивным методом

Попробуем просто сгруппировать объекты по материалу, не учитывая никаких дополнительных условий (рис. 3). Получим оптимальный порядок, так как все объекты, имеющие разделяемые состояния, идут в нем подряд, и состояние необходимо менять только один раз на всю группу. Так как каждым материалом нужно отрисовать хотя бы один объект, эта последовательность минимальна в смысле изменений состояния.

Рис. 4. Проблемы сортировки наивным методом

Однако, такая раскраска может нарушить порядок между объектами.

Дело в том, что объект, находящийся в *порядке наложения* (*z-order)* ниже другого, должен быть нарисован первым, так как второму нужно будет произвести альфа-смешивание с тем, над чем он находится (рис. 4).

3.1.2. Сохранение порядка

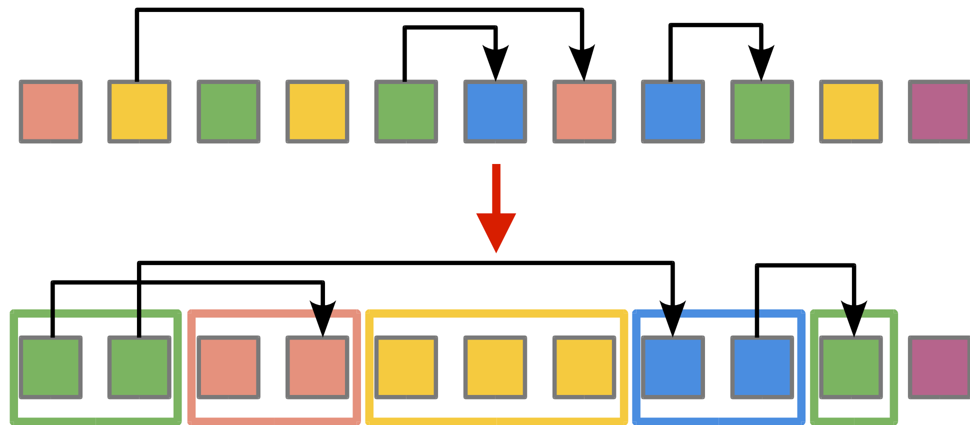

Рис. 5. Сортировка с сохранением порядка

Теперь все зависимости соблюдены (рис. 5). Однако «зеленая» группа оказалась разорвана. Тем не менее, это оптимальное упорядочение объектов с точки зрения смены состояний, сохраняющее связи между объектами.

3.2. Формальная постановка задачи

Экземпляр задачи состоит из множества *объектов*:

$$A = \{ \alpha_1, \alpha_2, \dots , \alpha_n \}$$

и *классов* (*материалов*):

$$M = \{ \mu_1, \mu_2, \dots, \mu_m \}.$$

Также задана функция сопоставления объекту его материала:

$$\varphi: A \to M.$$

И на множестве объектов задан частичный порядок:

$$\leq = D = \{ \langle \alpha_i, \beta_i \rangle \mid \alpha_i \in A, \beta_i \in A, i = 1..k \}^+.$$

Допустимыми решениями будут являться упорядочения множества A.

Целевой функцией оптимизации будет число индексов в упорядоченной

последовательности объектов, в которых происходит смена материала:

$$m(\alpha) = |\ \{\ i\ |\ \varphi(\alpha_i) \neq \varphi(\alpha_i + 1)\ \}\ |\ .$$

Таким образом, задача заключается в том, чтобы найти такой порядок объектов $\alpha_1, \alpha_2, \dots, \alpha_n$, что:

а) из зависимости $\alpha_i \leq \alpha_j$ следует $i < j$ (то есть соблюдается заданный частичный порядок),

б) и одновременно мощность множества точек смены материала минимально.

4. Анализ задачи

Мы покажем, что сформулированная задача принадлежит классы *NP-полных*. Для доказательства этого факта обратимся сначала к задаче о кратчайшей общей надпоследовательности и ее специальному случаю.

4.1. Основные определения

Следующие определения даны в соответствии с принятыми в [APPROX] и [NPC].

Задача *P* называется *задачей распознавания*, если множество экземпляров задачи разделено на два непересекающихся подмножества: положительные экземпляры и отрицательные, а задача состоит в определении, принадлежит ли данный экземпляр к множеству положительных.

Задача распознавания *P* *решается алгоритмом A*, если приняв на вход надлежащим образом закодированный экземпляр задачи I_P, алгоритм *A* завершается, и его результатом является ответ «ДА» тогда и только тогда, когда I_p принадлежит множеству положительных экземпляров.

Задача распознавания *P* решается *недетерменированным* алгоритмом *A*, если для любого экземпляра задачи I_P и любой возможной последовательности недерменированных выборов (угадываний) алгоритм завершается, и I_p принадлежит множеству положительных экземпляров тогда и только тогда, когда существует такая последовательность «угадываний», что результатом алгоритма является ответ «ДА».

Задача принадлежит *классу NP*, если существует недетерменированный алгоритм, ее решающий, который для любого экземпляра задачи завершается за время $O(|I_P|^k)$ для некоторого *k*.

Задача распознавания P_1 *сводится по Карпу* к задаче P_2 ($P_1 \leq_m P_2$), если

существует алгоритм R, который получив на вход экземпляр задачи P_1 I_{P_1}, преобразует его к экземпляру задачи P_2 I_{P_2}, что I_{P_1} является положительным экземпляром тогда и только тогда, когда I_{P_2} — положительный экземпляр.

P_1 называется *эквивалентной по Карпу* задаче P_2, если существуют сведения $P_1 \leq_m P_2$ и $P_2 \leq_m P_1$.

Сведение называется *полиномиальным*, если алгоритм R, выполняющий сведение, завершается за полиномиальное время.

Задача P называется *NP-трудной*, если для любой задачи P' из NP существует полиномиальное сведение по Карпу к P.

Задача называется *NP-полной*, если она NP-трудная и принадлежит NP.

Задача оптимизации P представляет собой четверку $\langle I_P, SOL, m, goal \rangle$, где I — множество экземпляров задачи, $SOL(i)$ — функция, сопоставляющая экземпляру задачи множество допустимых решений, $m(i, s)$ — целевая функция для оптимизации, а $goal$ — направление оптимизации (min, max).

Соответствующей задачей распознавания для задачи оптимизации $P = \langle I_P, SOL, m, goal \rangle$ называется задача P_D, решающая для экземпляра задачи оптимизации $I \in I_P$ и числа k, существует ли допустимое решение, оцениваемое целевой функцией не больше k (не меньше k в случае $goal = \max$): $\exists e \in SOL(I), m(I, e) < k$.

Задача оптимизации $P = \langle I_P, SOL, m, goal \rangle$ называется *NPO-задачей оптимизации*, если множество экземпляров I полиномиально распознаваемо, размер допустимых решений ограничен полиномом от размера экземпляра, а целевая функция вычислима за полиномиальное время. Такое определение обосновано тем, что соответствующая задача распознавания окажется в NP (теорема, см. [APPROX], 1.4).

Задача оптимизации называется *NPO-трудной*, если соответствующая ей задача распознавания является NP-полной.

Оракулом для задачи P является абстрактное устройство, которое предо-

ставляет ответ для экземпляра задачи P за один вычислительный шаг.

Говорят, что задача P_1 (не обязательно задача распознавания) *сводима по Тьюрингу* к задаче P_2 ($P_1 \leq_T P_2$), если существует алгоритм решения P_1, который может обращаться к оракулу для P_2.

Если *NPO*-трудная задача P_1 сводится по Тьюрингу к задаче P_2, то P_2 тоже является *NPO*-трудной (теорема, см. [NPC], 5.1).

4.2. Задача о кратчайшей общей надпоследовательности

Надпоследовательностью данной последовательности s длины m называется последовательность S, если существует последовательность индексов $j_1, j_2, \dots, j_m$, что для всех i из $1..m$ $s_i = S_{j_i}$.

Экземляр задачи о *кратчайшей общей надпоследовательности* (*shortest common supersequence, SCS*) состоит из множества $\{ s_1, s_2, \dots s_n \}$ последовательностей символов фиксированного алфавита $\sum$. Необходимо найти последовательность S наименьшей длины, что S является надпоследовательностью для всех s_i.

Более формально, в качестве задачи оптимизации $\langle I, SOL(i), m(i,s), goal \rangle$ она состоит из следующих компонент:

- множество экземпляров задачи I содержит коллекции последовательностей заданного алфавита $\sum$;
- допустимое множество для экземпляра $i = \{ s_1, s_2, \dots s_n \}$ $SOL(i)$ содержит все общие надпоследовательности для $s_1, s_2, \dots s_n$;
- целевая функция $m(s)$ это длина возможного решения: $m(s) = |s|$;
- критерий поиска goal = min.

Эта задача является *NP*-полной [SCS] и *W*[1]-трудной, будучи параметризованной числом исходных последовательностей [SCSFPT]. Также доказано, что задача остается *NP*-полной для случая бинарного алфавита [SCSBIN].

Вслед за работой [SCSFPT] определим полезные понятия для задачи.

Общую длину исходных строк $\sum_{i=1}^{n}|s_i|$ обозначим g. Назовем *размещением* для множества строк $M = \{ s_1, s_2, \dots s_n \}$ и их надпоследовательности S такую функцию $\varphi(i, j)$, что $S[\varphi(i, j)]=s_i^j$ и $\forall j, 1<j<|s_i|-1 \Rightarrow \phi(i, j)<\phi(i, j+1)$. Функцию $\gamma(j)=\{i | \exists k, \phi(i, k)=j\}$ назовем *источником* символа j. Важно, что $\sum_{j=1}^{|S|}|\gamma(j)|=g$, так как каждый символ исходных строк имеет свой образ в S.

4.3. Случай неповторяющихся символов

Рассмотрим такое ограничение задачи SCS, что в исходных последовательностях нет повторяющихся подряд символов. Для доказательства *NP*-полноты рассматриваемой задачи упорядочения нам потребуется показать, что такая подзадача SCS остается *NP*-трудной.

Рассмотрим соответствующую задачу распознавания. Дано целое число $k > 0$ и множество последовательностей $\{ s_1, s_2, \dots s_n \}$ символов алфавита $\sum$, что для любых i и j, $1 < i < n$, $1 < j <|s_i|-1$, выполняется $s_i^j \neq s_i^j+1$. Проверить, существует ли общая надпоследовательность для $s_1, s_2, \dots s_n$ длины не более k.

Покажем, что для любого экземпляра общей задачи SCS с множеством последовательностей $\{ s_1, s_2, \dots s_n \}$ и алфавитом $\sum$ можно построить такой экземпляр ограниченной задачи с множеством $\{ s_1', s_2', \dots s_n' \}$ и алфавитом $\sum'$, что ответ будет совпадать. Добавим в $\sum$ n символов-разделителей: $\sum' = \sum \cup \{d_1, d_2, \dots, d_n\}$. Пусть $s_i'^j$ будут построены из s_i^j добавлением соответствующего разделителя d_i после каждого символа. Очевидно, что условие отсутствия повторяющихся подряд символов выполняется, так как после каждого символа из $\sum$ идет разделитель. Заметим, что каждая s_i является подпоследовательностью s_i'. Также, пусть $k'=k+g$.

Пусть ответ для полученной задачи положительный. Следовательно, существует общая надпоследовательность S длины $m < k'$, которая, кроме всего прочего, содержит $\sum_{i=1}^{n}|s_i|$ символов-разделителей. Каждая s_i является подпоследовательнотью S (так как содержится в s_i') и не содержит разделителей. Раз-

делителей в S содержится ровно g штук, потому что они различны для каждой исходной строки. Поэтому, удалив все разделители получим общую надпоследовательность $s_1, s_2, \ldots s_n$ длины, не более $k'-g=k$.

Предположим наоборот, что ответ для полученной задачи отрицательный. Чтоб показать, что из этого следует отрицательный ответ исходной задачи, предположим, что для $s_1, s_2, \ldots s_n$ на самом деле существует надпоследовательность S, короче k. Расположим после каждого j-ого символа S разделители, соответствующие источникам $\gamma(j)$ этого символа. Обозначим эту расширенную строку S'. Длина S' не превосходит $k+\sum_{j=1}^{|S|}|\gamma(j)|=k+g=k'$. Одновременно, S' является надпоследовательностью $s_1', s_2', \ldots s_n'$, так как в S' можно выделить s_i', оставив в ней символы S, соответствовавшие s_i и разделители d_i. То есть, S' удовлетворяет условиям ограниченной задачи, ответ для нее должен был быть положительным. Из полученного противоречия заключим, что ответ для исходной задачи в этом случае был отрицательным.

Показанное сведение по Карпу показывает, что рассматриваемое сужение задачи SCS на последовательности без повторяющихся подряд символов является *NP*-трудным (и, естественно, *NP*-полным в силу принадлежности общей задачи к *NP*).

NP-полнота этой задачи означает *NPO*-трудность соответствующей задачи оптимизации (см. 4.1).

4.4. NP-полнота задачи упорядочения

В соответствии с теоремами параграфа 4.1 NP-полнота задачи распознавания, соответствующей некой *NP*-задаче оптимизации, эквивалентна NPO-трудности последней. Так как сведения по Тьюрингу обычно более наглядны, мы докажем *NPO*-трудность рассматриваемой задачи, предъявив сведение к ней ограниченной задачи SCS.

Несложно видеть, что сформулированная задача является *NP*-задачей оп-

тимизации, так как и допустимые решения (упорядочения) очевидно не превышают размера экземпляра задачи. Целевая функция, в свою очередь, вычисляется за линейное время от длины допустимого решения.

Для доказательства *NPO-трудности* рассмотрим сведение задачи о наименьшей общей надпоследовательности без повторяющихся символов. В предыдущем параграфе было доказано, что это сужение задачи является *NPO*-трудным.

Преобразуем экземпляр SCS в условие нашей задачи оптимизации (назовем его внутренней задачей). Пусть алфавит Σ будет множеством классов внутренней задачи. Каждому символу x_i^j исходной строки i поставим в соответствие объект, который для простоты будем называть так же — x_i^j. Функция φ будет отображать объект x_i^j в символ алфавита $\varphi(x_i^j) \in \Sigma$. Частичный порядок будет определен в соответствии с порядком следования символов в исходных строках. Таким образом получим внутреннюю задачу оптимизации.

Из решения полученной задачи r^u можно построить решение исходной задачи SCS. Преобразуем объекты полученной последовательности обратно в символы алфавита с помощью φ. Назовем полученную последовательность промежуточным ответом r^0. Затем для получения конечного ответа r достаточно всего лишь удалить последовательные повторы символов в строке.

Для примера рассмотрим экземляр SCS $\{ abc, dabd \}$. Оптимизация числа переходов между символами даст ответ $r^0 = daabbcd$. После удаления последовательных повторов получим строку $r = dabcd$, являющуюся правильным ответом задачи примера.

Покажем, что последовательность, полученная описанным образом, является правильным ответом для исходной задачи SCS. Для начала заметим, что промежуточный ответ является общей надпоследовательностью: все символы исходных строк в нем присутствуют, и по самому определению внутренней задачи порядок символов не нарушен. По определению надпоследовательности, исходную последовательность можно получить из нее посредством удаления в

ней некоторого числа символов. Заметим, что в процессе этого последовательные повторы символов неминуемо будут удалены, так как исходные строки не имели повторов. Таким образом, конечный ответ все еще является общей надпоследовательностью.

Конечный ответ также является оптимальным. Заметим, что в силу отсутствия повторов в строке, число изменений символа (позиций i, что $r_i \neq r_{i+1}$) равно длине строки минус один. Допустим r не является наименьшей общей надпоследовательностью. Тогда существует более короткий правильный ответ r', с еще меньшим числом смен символа. Выделим в r' подпоследовательности, соответствующие исходным строкам, и припишем каждому символу соответствующие объекты x_i^j (возможно несколько для каждого символа). Заменяя символы r' на последовательности приписанных объектов получим более оптимальный в смысле смены состояний ответ внутренней задачи. Это противоречит тому, что решение r^u оптимально для внутренней задачи.

Таким образом задача рассматриваемая задача оптимизации смен состояния не менее сложна, чем *NP*-полная задача поиска кратчайшей общей надпоследовательности. Поэтому является *NP*-трудной и *NP*-полной.

4.5. Связанные задачи

Сходные задачи возникают в разных областях исследований, в основном в формулировках, связанных с планированием производства (*Sequencing and Scheduling*). Похожие задачи также фигурируют в сборниках *NP*-полных задач ([COMPEN], [NPC]).

Чтобы было проще ориентироваться в соответствующей литературе, переформулируем задачу на язык, обычно используемый в задачах планирования производства. В нашем случае есть универсальный станок с магазином из m инструментов. Также задано n производственных действий, для которых известен инструмент из магазина, который предназначен для данного действия. Частичный порядок при этом возникает естественным образом: окраска и лакировка деталей, например, должна производиться после выпиливания их из бруска

древесины. Станок выполняет действия последовательно, при этом если для следующего действия требуется другой инструмент, нежели для предыдущего, станок тратит время на смену инструмента из магазина. Задача заключается планировании порядка выполнения задач, чтоб минимизировать затраты времени на смену инструмента.

Большинство работ не ставят задачей учет порядка выполнения действий. Например в работе [MAINT] минимизируется количество запаздывающих задач с учетом периодических простоев для обслуживания.

Другие работы ([AGNETIS1], [AGNETIS2]) исследуют оптимальное разделение на пакеты для параллельной обработки. При этом рассматриваемая в данной работе задача упорядочения действий для минимизации смен инструмента возникает в качестве подзадачи. Однако в этих работах она была решена с жесткими ограничениями: $n = 3$ или $m = 2$.

Одна из задач статей [AGRAWAL] и [RAO] посвящена минимизации единиц хранения промежуточных результатов между действиями. То есть ребрам графа сравнимости частичного порядка присвоены веса, и необходимо минимизировать взвешенную разницу времени старта пар действий. Эта задача похожа на нашу тем, что «связанные» действия (те, между которыми нужно сохранять много промежуточных результатов) требуется располагать как можно ближе в упорядочении. Однако результаты, полученные авторами, не удается непосредственно применить к нашему случаю.

И наконец K.H. Ecker и J.N.D. Gupta в работе [ECKER] рассматривают строгое обобщение нашей задачи. Вместо соответствия действиям инструментов и минимизации смен разных инструментов, они предполагают, что каждое действие требует своего уникального инструмента. Но для каждой пары инструментов определена длительность переключения. Требуется минимизировать общее время, затраченное на смену инструментов.

В работе приведена неполная схема доказательства NP-полноты задачи. Этот недостаток мы надеемся восполнить данной работой. Основная идея доказательства, то есть использование задачи SCS, сохраняется. Однако повторяющиеся символы в экземплярах этой задачи требуют аккуратной обработки.

Ecker и Gupta представляют два алгоритма решения своей задачи. Первый основан на идее динамического программирования и находит точное решение за-

дачи, но, естественно, требует экспоненциального времени работы и, к тому же, экспоненциального количества памяти для хранения графа подзадач.

Второй алгоритм является приближенным алгоритмом. Он основан на произвольном выборе нескольких допустимых решений и построении на их основе подграфа подзадач, аналогичного графу подзадач в динамическом алгоритме поиска точного решения. На основе такого подграфа можно построить приближенное решение задачи.

Такая эвристика оказывается весьма эффективной. Авторы не приводят жестких оценок точности для их аппроксимации (и, скорее всего, удовлетворительные теоретические оценки для этого метода получить очень трудно). Несмотря на это, экспериментальная проверка показывает, что при начальной генерации 10-20 допустимых решений точность составляет от 88 до 99 процентов.

Возможные улучшения этого алгоритма предполагают эффективный выбор начальных решений. Важно, что улучшенный жадный алгоритм, рассматриваемый ниже, может быть использован для начальной генерации и, таким образом, может улучшить результаты авторов.

5. Предлагаемый алгоритм

5.1. Требования

Тестирование выявило, что на практике для отрисовки экранов оптимизируемого приложения используется порядка N = 100..300 объектов. При этом алгоритм должен выполняться во время отображения каждого кадра. Исходя из требуемой частоты кадров в 60 Гц, получим максимальное время работы в 1/60 с = 16 мс. Поэтому надо выбрать простой и быстрый алгоритм, который является не только асимптотически эффективным, но и обеспечивает небольшие константные множители сложности. Мы рассмотрим улучшенный жадный алгоритм, аппроксимирующий решение задачи.

5.2. Специальные случаи

Очевидно, что для практического применения следует выбрать алгоритм не только асимптотически эффективный в общем случае, но тот, который работает с реально возникающими экземплярами задачи за разумное абсолютное время. Поэтому посмотрим, какими свойствами обладают структуры частичного порядка интерфейсов.

Графом сравнимости ([GRAPHS]) для частичного порядка называется неориентированный граф, у которого множеством вершин является носитель частичного порядка, а ребром соединены объекты a и b, для которых известно, что $a \leq b$ либо $b \leq a$. Так вот, граф сравнимости для частичного порядка, порожденного пересекающимися прямоугольниками на плоскости (видимые объекты на экране) имеет свойство *прямоугольниковости* (*boxicity*) равное двум (см. [BOX]). Такие графы также называются *графами пересечения прямоугольников* (*rectangle intersection graphs*). Известно, что множество таких графов сов-

падает с множеством *внешнепланарных* (*outerplanar*) графов, то есть таких планарных графов, у которых каждая вершина лежит на внешней границе плоского вида графа. Многие *NP*-полные задачи, такие как задача о клике, раскраске вершин, вершинном покрытии, в сужении на множество таких графов имеют эффективные решения или схемы полиномиальной аппроксимации ([RIM], [YEHUDA]).

Однако наша задача остается *NPO*-трудной даже в этом случае. Легко видеть, что граф сравнимости частичного порядка, используемый в доказательстве NPO-трудности, был графом пересечения прямоугольников. Частичный порядок в этом сведении был объединением нескольких полных порядков, определенных исходными последовательностями. А граф полного порядка можно представить как последовательность вложенных друг в друга прямоугольников.

Зато другая особенность частичного порядка объектов интерфейса позволит нам показать эффективность предлагаемого метода для практически важных случаев. Рассмотрим *ярусно-параллельную форму* графа *(рис. 6)* частичного порядка объектов. Ярусно-параллельная форма графа — это «деление вершин ориентированного ациклического графа на перенумерованные подмножества V_i такие, что, если дуга e идет от вершины $u \in V_j$ к вершине $v \in V_k$, то обязательно $j < k$» [LAYERS].

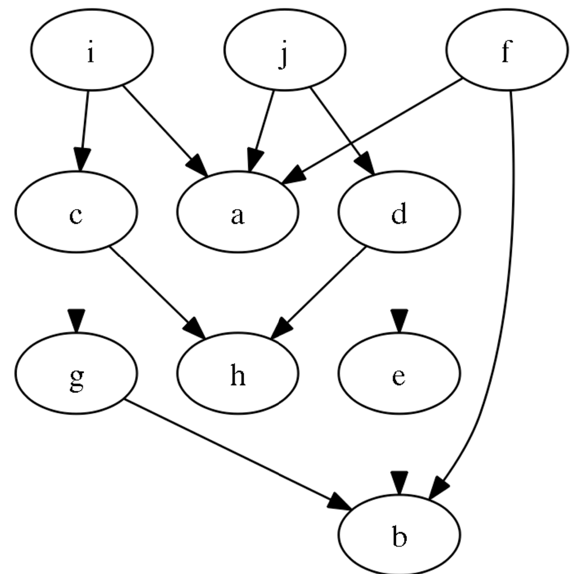

Рис. 6. Ярусно-параллельная форма графа

В графах сцен интерфейсов часто встречаются поддеревья сходной структуры. Например, интерфейс списка приложений содержит много блоков, состоящих из пиктограммы и названия приложений. Из-за существования схожих поддеревьев многие классы объектов будут локализованы на отдельном уровне ярусно-параллельной формы графа сравнимости. Эта особенность пона-

добится в дальнейших рассуждениях.

5.3. Псевдо-код

1. Для каждого элемента проинициализировать $d[\alpha]$ — число входящих связей.

   $d[\alpha] \leftarrow | \{ \alpha_i \mid \langle \alpha_i, \alpha \rangle \in D \} |$.

2. Инициализировать $g[\mu]$ числом элементов в группе, соответствующей классу μ.

   $g[\mu] \leftarrow | \{ \alpha_i \mid \varphi(\alpha_i) = \mu \} |$.

3. Сгруппировать независимые элементы ($d[\alpha] = 0$) по материалам в семейство групп S.

   $S \leftarrow \{ \langle \mu_i, E_i \rangle \mid i = 1..m, E = \{ \alpha_i \mid \varphi(a_i) = \mu, d[\alpha_i] = 0 \} \}$.

4. Выбрать группу G, которая содержит все необработанные объекты соответствующего материала: $|G| = g[\mu]$.

   Если такой не окажется, выбрать наибольшую по мощности группу G:

   $\langle \mu, G \rangle \in S, |G| = \max |E_i|$.

5. Для каждого члена группы $\alpha \in G$:

   a) Для каждого элемента β, зависящего от α ($\langle \alpha, \beta \rangle \in D$):

      i. $d[\beta] \leftarrow d[\beta] - 1$.

      ii. Если $d[\beta] = 0$, добавить его в группу, соответствующую его материалу $\varphi(\beta)$: $G \leftarrow G \cup \{\beta\}$.

6. Выдать G на выход алгоритма.

7. Удалить $\langle \mu, G \rangle$ из S.

   $S \leftarrow S \setminus \{ \langle \mu, G \rangle \}$.

8. $g[\mu] \leftarrow g[\mu] - |G|$.

9. Если S не пусто, перейти к шагу 4.

5.4. Анализ

Предложенный алгоритм соблюдает условие сохранения порядка, так как он выбирает объект α, только после того, как все объекты, от которых он зависит, были выбраны (см. условие $d[\alpha] = 0$ на шаге 5.a.i). В силу критериев выбора максимальной группы G на шаге 4 результат работы алгоритма лучше, чем случайный порядок.

Сложность предложенного алгоритма $O(n \log m)$. Каждый объект задачи рассматривается один раз (в циклах на шагах 5. и a.). При правильной организации структур данных обновление семейства групп на шаге 5.a.ii требует времени, пропорционального логарифму числа материалов, а выбор лучшей группы в этом случае произойдет за константное время.

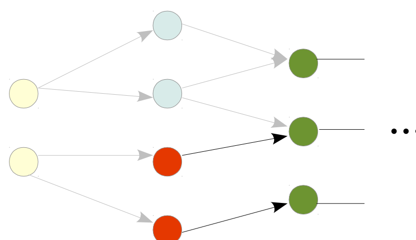

Рис. 7. Ярусно-параллельная форма графа зависимостей на третьей итерации

Покажем, в каких случаях предложенный алгоритм будет выдавать оптимальный результат. Рассмотрим *ярусно-параллельную форму* графа частичного порядка объектов.

Если множества материалов, используемых на каждом уровне, не пересекаются, то полученная с помощью предложенного алгоритма последовательность будет оптимальна. В самом деле, в начале работы алгоритма в группах S находятся элементы первого уровня графа: на первом уровне находятся вершины, в которые не входит ни одно ребро (независимые элементы). Пусть на некотором итерации k верно, что на итерациях $1..k{-}1$ в шаге 4 группы выбирались по первому условию, то есть выбиралась группа, содержащая все элементы своего

материала (см. рис. 7). Тогда для некоторого материала μ на первом уровне дерева, содержащем необработанные элементы, есть все элементы этого материала. Поэтому на итерации k первое условие шага 4 выполнится, и будут выбраны для обработки все элементы некоторого материала. По индукции получим, что на каждой итерации в выходную последовательность будут добавляться элементы целыми классами. При этом число смен материала в выходной последовательности составит $m - 1$. Такая последовательность очевидно является полностью оптимальной.

Надо заметить, что рассмотренный случай практически является очень важным, потому графы сцен интерфейсов имеют тенденцию располагать объекты одного класса на одном уровне. Такие случаи и являлись главной мишенью нашей оптимизации. Описанный алгоритм позволяет быстро и эффективно группировать такие структуры.

6. Тестирование

6.1. Выигрыши, обусловленные объединением геометрий

Результаты тестирования согласуются с тестами NVIDIA, описанными в [BATCH]. На рис. 8 и 9 показаны графики зависимости производительности обработки треугольников от размера пакета (шкала по оси x логарифмическая). Данные, использованные для построения графиков усреднены по шестидесяти замерам. Видно, что с увеличением размера пакета производительность логарифмически растет. Однако, при увеличении размера пакета сверх 256 треугольников, можно видеть различия в поведении разных графических адаптеров. Для некоторых наблюдается снижение производительности, у других — стабилизация. Эти эффекты можно объяснить влияние других факторов, например особенностями кеширования и превышением объема внутренней памяти адаптера. Эти факторы обычно достаточно специфичны для конкретных моделей устройств.

6.2. Выигрыши, обусловленные группировкой по материалам

В реальном приложении для сложных сцен применение предложенного алгоритма позволило добиться увеличения частоты кадров с 20-30 Гц до 55-60.

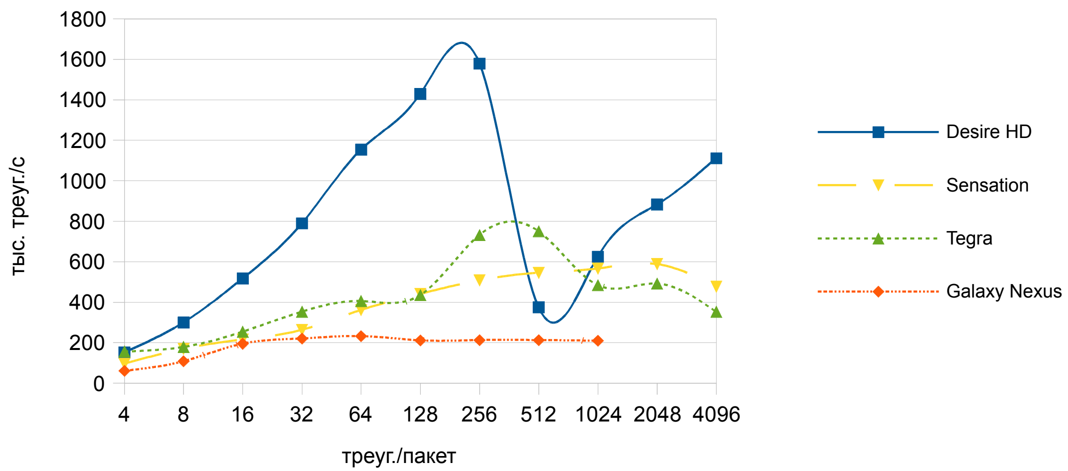

Рис. 8. Зависимость числа обрабатываемых треугольников в секунду от размера пакета

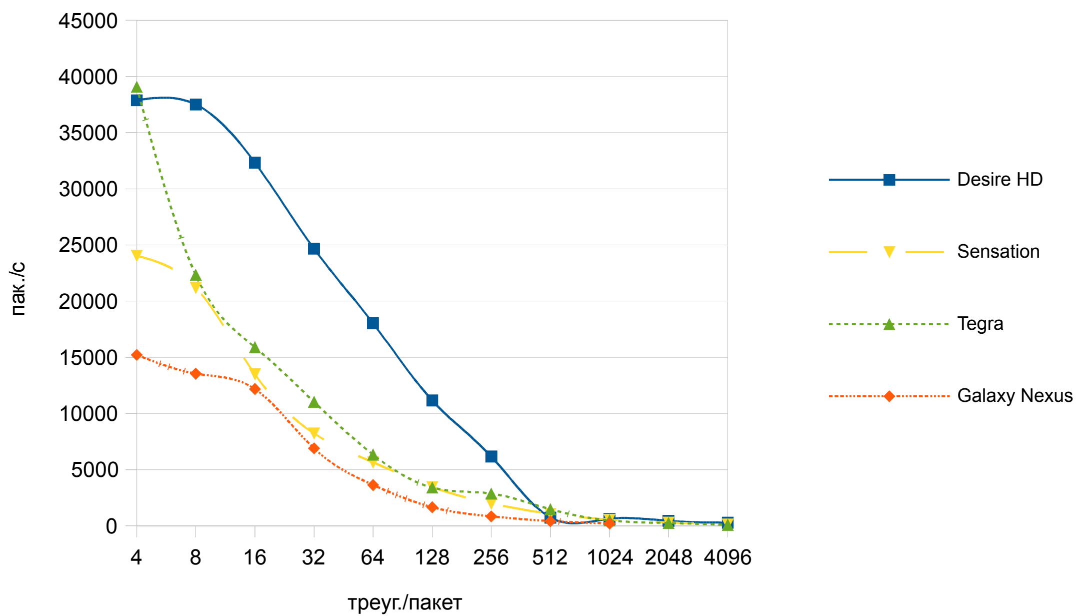

Рис. 9. Зависимость числа обрабатываемых пакетов в секунду от размера пакета

7. Заключение

Методы оптимизации рассмотренные в данной работе были успешно интегрированы в существующий проект и привели к значительному увеличению производительности интерфейса. Также это позволило использовать в приложении более богатую и привлекательную графику.

Кроме того, в работе была доказана *NP*-полнота задачи упорядочения объектов, максимизирующего локальность и сохраняющей частичный порядок. Для этой задачи предложен и протестирован эвристический алгоритм решения, позволяющий улучшить свойство локальности, сохраняя порядок.

Этот же алгоритм может быть использован для улучшения характеристик другого алгоритма решения схожей задачи.

В качестве материала для дальнейшего исследования можно рассмотреть другие подходы к аппроксимации описанной *NP*-трудной задачи.

8. Источники

[SHELL] SPB Shell 3D, http://spb.com/android-software/shell/

[IPHONE] Apple Sold More iPads In Q4 Than Any Single Manufacturer Sold PC Devices, http://techcrunch.com/2012/03/07/apple-sold-more-ipads-in-q4-than-any-single-pc-manufacturer/

[GRAPH] *Bar-Zeev, Avi*: Scenegraphs: Past, Present, and Future / http://www.realityprime.com/articles/scenegraphs-past-present-and-future. 2007

[IRRLICHT] Irrlicht 3D, http://irrlicht.sourceforge.net/

[LINDERDAUM] Linderdaum Engine, http://www.linderdaum.com/

[COCOS] Cocos 2d for iPhone, http://www.cocos2d-iphone.org/

[OGRE] Ogre 3d, http://www.ogre3d.org/

[TRAPP] *Trapp M.*: OpenGL-Performance and Bottlenecks / Analyse, Planung und Konstruktion Computergrafischer Systeme seminar. 2003/2004

[ATLAS] *NVIDIA*: Improve Batching Using Texture Atlases / SDK White Paper. 2004

[ORDER] *Sander P. V., Nehab D., Barczak J.*: Fast Triangle Reordering for Vertex Locality and Reduced Overdraw / ACM Transactions on Graphics. 2007

[CULL] *Nehab D., Barczak J., Sander P. V.*. Triangle Order Optimization for Graphics Hardware Computation Culling / ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 2006

[BATCH] *Wloka M.*. Batch, Batch, Batch: What Does It Really Mean? / NVIDIA GameDevelopers conference 1. 2003

[APPROX] *Ausiello G., Crescenzi P., Kann V., Marchetti-Spaccamela A., Gambosi G.*: Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties. Springer, 2003

[NPC] *Garey M. R.*: Computer and intractability. A guide to the theory of NP-

completeness. Bell Labs, 1979

[SCS] *Maier D.*: The Complexity of Some Problems on Subsequences and Supersequences / Journal of the Association for Computing Machinery, Vol 25, No 2. 1978

[SCSFPT] *Pietrzak K.*: On the Parameterized Complexity of the fixed alphabet Shortest Common Supersequence and Longest Common Subsequence Problems / Journal of Computer and System Sciences. 2002

[SCSBIN] *Raiha, K.-J., Ukkonen, E.*: The shortest common supersequence problem over bi-nary alphabet is NP-complete / Theretical Computer Science. 1981

[COMPEN] *Crescenzi P., Kann V.*: A compendium of NP optimization problems / ACM SIGACT News. 2000

[MAINT] *Lee J., Kim Y.*: Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance / Computers & Operations Research. 2011

[AGNETIS1] *Agnetis A., Alfieri A., Nicosia G.*: Part Batching and Scheduling in a Flexible Cell to Minimize Setup Costs / Journal of Scheduling. 2003

[AGNETIS2] *Agnetis A., Alfieri A., Nicosia G.*: A heuristic approach to batching and scheduling a single machine to minimize setup costs / Computers & Industrial Engineering. 2004

[AGRAWAL] *Agrawal A., Klein P., Ravi R.*: Ordering Problems Approximated: Register Sufficiency, Single-Processor Scheduling and Interval Graph Completion / Technical report, Brown University. 1991

[RAO] *Rao S., Richa A.*: New Approximation Techniques for Some Ordering Problems / Proceedings of Ninth Annual ACM SIGAM Symposium on Discrete Algorithms. 1998

[ECKER] *Ecker K. H., Gupta J.N.D.*: Scheduling tasks on a flexible manufacturing machine to minimize tool change delays / European Journal of Operational Research. 2004

[GRAPHS] *Brandstädt A., Le V. B., Spinrad J. P.*: Graph Classes. A Survey. Society for Industrial and Applied Mathematics, 1999

[BOX] *Adiga A., Bhowmick D., Chandran L. S.*: Boxicity and Poset Dimension / Computing and Combinatorics. 2010

[RIM] *Rim C. S., Nakajima K.*: Complexity Results for Rectangle Intersection and Overlap Graphs / Institute for Systems Research Technical Reports. 1988

[YEHUDA] *Bar-Yehuda R., Hermelin D., Rawitz D.*: Minimum Vertex Cover in Rectangle Graphs / Computational Geometry. 2011

[LAYERS] *Healy P., Nikolov N.*: How to Layer a Directed Acyclic Graph / Lecture Notes in Computer Science, Vol 2265. 2002