Алгоритм компрессии воксельного поля, оптимизированный для хранения данных о ландшафте (ANIRLE)

Л.М.Л. Бланко, С.А. Березняк, П.А. Пантелюк

Введение

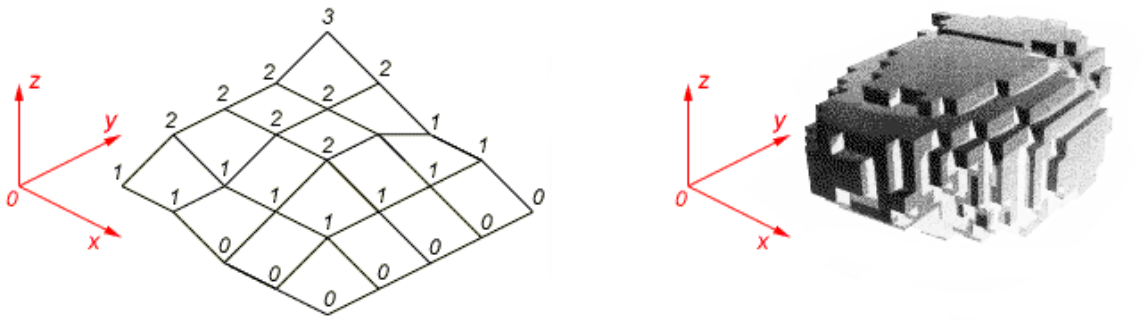
Трехмерное моделирование различных объектов представляет большие преимущества в процессе их исследования и проектирования. С этой целью создается множество программных комплексов различной направленности [1, 2]. В настоящее время всё большее распространение получает способ представления данных о форме и содержимом объектов с помощью воксельной информации. Использование вокселей – трёхмерных аналогов пикселей – всегда было сопряжено с чрезмерными затратами памяти, как дисковой, так и оперативной. Однако хранение объёмных данных таким образом имеет и неоспоримые плюсы, с чем и связана растущая популярность этого подхода. Во-первых, в таком виде может быть сохранён объект абсолютно произвольной формы. Во-вторых, сохраняется информация о внутреннем строении объекта, что даёт преимущества в моделированиинепрерывных сред и взаимодействий с этими моделями [3, 4].

Наиболее заметное применение воксельные данные получили в сферах проектирования, медицины, и, как ни странно, компьютерных игр и развлечений. Хотя, единичные разработки существовали уже довольно давно, "взрыв" популярности воксельных ландшафтов в игровых проектах произошёл буквально пару лет назад с появлением проекта Minecraft. Ключевым преимуществом для пользователей стала как раз возможность исследовать процедурно генерируемые игрой пейзажи и редактировать их в процессе игры.

Авторами статьи был предложен алгоритм хранения и доступа к данным, ключевым назначением которого является именно снижение затрат памяти. Рассматривается случай реалистичной модели рельефа, имеющей большие объёмы однородного материала. Алгоритм предназначен для задач восстановления полигональной поверхности, где потребуется полный, но относительно последовательный доступ к случайным участкам воксельных данных внутри всего объёма (как в работе [5]).

Анализ существующих работ

Сравнительно долго в представлении открытых пространств доминировала техника карт высот. Используя двухмерные изображения, каждой точке пространства могла быть задана высота рельефа в этом месте. Этот простой подход имеет очевидные ограничения, так, например, нельзя описать отвесные или нависающие скалы, пещеры, крупные валуны.



(a) (b)

Рис. 1. –Карта высот (a) и воксельные данные (b).

В противоположность этому, как видно на рис.1, воксельные данные полностью описывают состояние каждой точки внутри заданного объёма. Соответственно, могут быть сохранены любые желаемые детали, как формы,

так и содержимого. Но всё имеет свою цену. Во-первых, визуализация таких данных является нетривиальной задачей и отдельным предметом множества исследований. Во-вторых, как уже упоминалось, возникают проблемы из-за больших требований к памяти компьютера. Дело в том, что карту высот мы можем рассматривать как одно единственное изображение, тогда как воксельные данные представляют собой N таких изображений, где N - максимальная высота моделируемого объёма.

Чтобы в ряде случаев уменьшить как вычислительную нагрузку, так и количество занимаемой памяти, предлагаются различные решения, зачастую основанные на технологии SVO (*Разреженное воксельное октодерево*). Однако, данная технология в первую очередь предназначена для решения проблем производительности, особенно для приложений, использующих трассировку лучей. Поэтому требования к памяти остаются весьма значительными, что причисляется к одному из недостатков технологии [3, 4, 6].

Кроме того, иногда используют и аналитические методы моделирования поверхностей: комбинируя различные математические выражения, можно описать практически любую поверхность. Этот подход очень эффективен, с точки зрения затрат памяти, так как вседанные о модели хранятся в виде относительно компактного аналитического описания [2]. В то же время, вычислительная сложность не позволяет применить его для приложений, работающих в реальном времени и для моделирования произвольных интерактивных ландшафтов. По этой причине предпочтение было отдано использованию воксельных данных, совместно с алгоритмом их компрессии, позволяющим оптимизировать затраты памяти.

Применение существующих алгоритмов компрессии, таких как Deflate [7, 8] оказалось неэффективным: они удобны для хранения данных на жестком диске с последующим извлечением содержимого архива перед работой.

Специфика нашей задачи создает необходимость отказаться от декомпрессии в принципе: необходимо непрерывно держать данные в сжатом состоянии, но чтобы при этом были возможны операции чтения и записи.

В данной статье предлагается алгоритм произвольного доступа к сжатым данным без необходимости их декомпрессии, основанный на алгоритме группового кодирования. Его эффективность в заданной предметной области позволяет с минимальными издержками уйти от проблем перегрузки памяти вычислительной системы при использовании воксельных данных для моделирования реалистичных ландшафтов.

1 Структура представления данных

1.1 Групповое кодирование

Так как основой данной технологии является групповое кодирование (англ. Run-LengthEncoding, RLE), первичными элементами данных становятся счётчики и блоки не-избыточных данных. В самом базовом алгоритме RLEединственный вид таких данных - один символ. В результате сжатые данные хранятся в формате: {<счётчик><символ>}, где счётчик - натуральное число, указывающее количество повторений символа. Очевидно, что отсутствие в исходных данных групп одинаковых символов приведёт к наихудшему результату компрессии: результат будет превышать исходный размер в два раза.

Дальнейшее развитие RLE вводит понятие пропускаемых блоков. При данном подходе отрицательные значения счётчика указывают на необходимое число пропусков. Так можно ликвидировать описанный выше недостаток, сведя потери к минимуму (1 лишний символ-счётчик на каждые 128 символов)[9, 10]. Именно такой подход подразумевается при дальнейших отсылках к RLE в этой статье.

1.2 Адаптивный размер блока

Тогда как стандартный подход RLE описывает использование одного байта под хранение счётчика, этого количества может быть объективно недостаточно в ряде ситуаций. Например, при наличии очень больших блоков повторяющихся символов, хотелось бы сразу указать как можно больший размер блока. Для крупных воксельных миров вполне характерной является ситуация, когда один и тот же символ повторяется миллионы раз. С другой стороны, постоянное использование большого количества памяти под счётчики сильно снизит производительность алгоритма компрессии на более разнообразных входных последовательностях, которые тоже часто встречаются в воксельных мирах, особенно вблизи поверхности земли.

Для того чтобы избежать избыточных накладных расходов и, в то же время, иметь возможность одинаково эффективно обеспечивать компрессию, как малых, так и крупных блоков данных, было решено ввести адаптивный элемент заголовка компрессии. Иными словами, был добавлен специальный флаг, состояние которого показывает, сколько байт использовано под счётчик. Для малого размера заголовка было решено использовать 1 байт, отводя 6 младших бит под счётчик, а два старших - под флаги пропуска и размера.

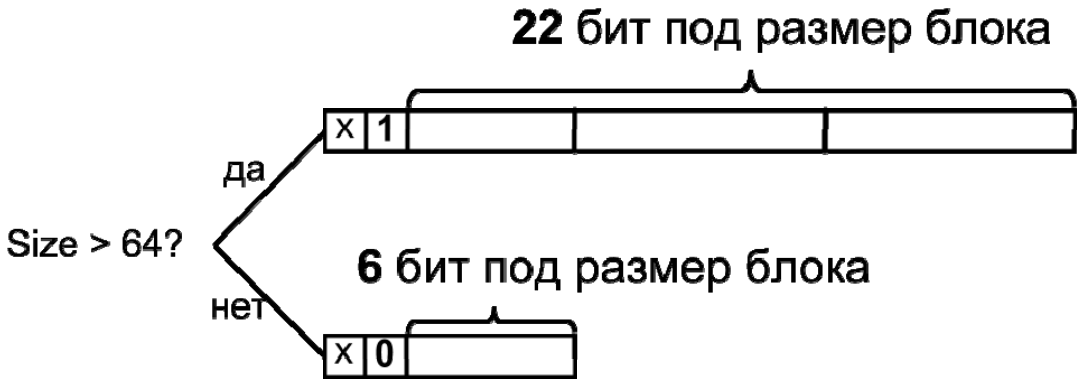


Рис. 2. – Условный выбор размера заголовка компрессии

Из рис.2 видно, что для крупного заголовка авторами статьи был выбран размер 3 байта, из которых 22 бита отводятся под счётчик, что позволяет описывать последовательности длиной до двух миллионов символов (2^{22}).

1.3“Не-чередование” данных

Как правило, данные, кодируемые RLE, содержат счётчики и сжатые данные "вперемешку", то есть от начала и до конца файла повторяется структура: {<счётчик><символ(ы)>}. Такой подход действительно удобен для хранения минимальной необходимой информации на носителях. Однако если мы ставим задачу произвольного редактирования сжатых данных, потребуется иное решение, на этапе когда данные уже загружены в оперативную память для работы. Основная проблема – это операция "вставить в середину" при

модификации данных. Так как взаимодействие происходит со сжатыми данными, результат модификации может серьёзно отразиться на структуре блоков. Так изменение одного символа в середине сжатого блока приведёт к разделению блока на 2 сжатых и 1 пропускаемый, то есть на месте одного блока возникает сразу три.

В рамках работы над алгоритмом, было решено содержать набор данных в оперативной памяти компьютера в виде двух отдельных списков. Первый список отвечает за заголовок компрессии, каждый его элемент – число, длина описываемого блока данных. Как уже говорилось, счётчик имеет два различных состояния для возможности описывать как сжатые, так и пропущенные данные. Второй список хранит непосредственно сжатые данные. Каждый его элемент - массив из одного или нескольких символов. Оба списка должны всегда содержать равное число элементов.Каждый блок данных непременно описывается заголовком, как и не может быть заголовка, не описывающего какой-либо существующий блок данных. Этот нюанс отражён и в названииANIRLE: AdaptiveNon-Interleaved (англ."не-чередующийся", "не-перемешанный")RLE.

1.4Структура исходных данных о ландшафте

Для описания ландшафтов, как правило, необходимо хранить данные нескольких типов: например, плотность вокселя и его материал. Первое служит для выражения формы объекта, а второе – его цвета (например, номериспользуемой текстуры)[5]. Эти два параметра имеют различный физический смысл, поэтому их значения совпадают крайне редко. В результате имеет смысл осуществлять компрессию этих данных двумя отдельными файлами, каждый из которых будет содержать значения только одного типа.

2. Алгоритмы работы с данными

2.1. Чтение данных

Одной из важнейших задач, решаемых описываемым подходом, является чтение данных без дополнительных затрат памяти. Простота используемого формата помогает свести такое чтение к набору простейших операций.

Рис. 3.– Блок-схема алгоритма чтения в буфер

Из рис.3, где изображена блок-схема общего подхода к чтению сжатых данных, видно, что этот процесс довольно прост в реализации.

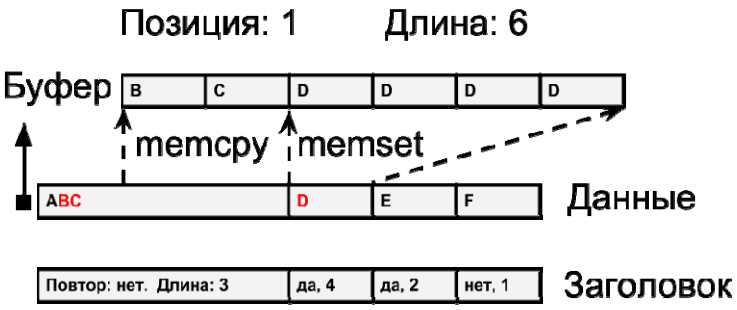


Рис.4. –Чтение данных в буфер

На рис.4 более детально представлен этот процесс на примере чтения нескольких байт. Допустим, требуется считать 6 байт начиная с байта номер 1. Судя по заголовку, начало буфера пересекается с данными первого блока, который оказался несжатым. В этом случае мы определяем, какую область следует копировать из блока, и затем копируем её стандартной функцией (memcpy, язык C). Следующий блок сжат, то есть в области данных он представлен единственным символом. Этот символ дублируется на всём протяжении блока. В соответствии с заголовком, этот блок содержит 4 символа, как раз те, что заполнят оставшиеся 4 байта буфера. В данном случае мы используем другую стандартную функцию (memset, язык C), чтобы задать значения всех следующих байтов буфера равными дублированному символу второго блока данных.

Процесс перехода к произвольной позиции внутри читаемых данных проще всего организовать линейно – простым перебором блоков в направлении искомой позиции, до обнаруженияперекрывающего её блока. Для повышения производительности может быть организован механизм бинарного поиска, существенно снижающий временные затраты на доступ к произвольной позиции.

2.2. Модификация данных

Перезапись произвольных участков файла является задачей, гораздо менее тривиальной, чем чтение. Причиной тому является высокая вероятность изменений в блочной структуре сжатого файла. Например, может потребоваться модифицировать один символ посреди большого блока повторяющихся значений. Очевидно, что это приведёт к разбиению старого блока на три части: два меньших участка с повторениями и один с новым, несовпадающим символом. Можно привести и множество других примеров, когда изменение лишь одного символа повлечёт крупные изменения в структуре сжатых данных.

Для преодоления обозначенных проблем, в общем случае, существует два способа. Во-первых, можно извлечь из сжатого файла все блоки, на которые способна повлиять вносимая модификация, после чего внести изменения и снова произвести компрессию по стандартному алгоритму. Недостаток этого подхода в том, что длины некоторых блоков могут быть весьма существенны, а значит, работа с ними будет требовать излишнего выделения памяти, а также вычислительной мощности на повторную компрессию.

Второй способ состоит в более разумном подходе к управлению сжатыми блоками. При модификации некоторого символа, придется определить какие блоки и как именно это затронет, после чего предпринять непосредственные шаги для достижения запланированного результата. Для этого требуется сначала произвести компрессию новых данных, чтобы они приняли единый вид с существующими сжатыми блоками, в набор которых они будут вставляться. Например, в случае модификации одного символа, новый записываемый символ будет представлен как блок неповторяющихся символов длиной 1 байт. При вставке новых блоков в существующую блочную структуру, необходимо определить какие изменения повлечет за собой контакт крайних блоков модифицируемых данных. Эти изменения могут быть выражены созданием и удалением блоков, модификацией их содержимого и длины. Таким образом, второй способ является явно более сложным в реализации. С другой стороны к его плюсам можно отнести отсутствие излишних затрат памяти и процессорного времени, что делает его более перспективным в целом.

Заключение

По результатам тестирования программного прототипа реализации описанных алгоритмов, можно сделать вывод о том, что описанный подход действительно помогает существенно уменьшить объёмы занимаемой памяти

при работе с воксельными данными. В качестве эксперимента, для сжатия были выбраны два набора данных: первый – пустое пространство, а второй – описание ландшафта вблизи поверхности. Пустое (однородное) пространство можно встретить довольно часто: высоко над землей и глубоко под ней, зачастую, не требуется детальное моделирование различных материалов. Пространство вблизи поверхности, напротив, встречается реже, но представляет наибольшую нагрузку для описанного подхода к сжатию данных. Для случая компрессии пустого пространства, как и ожидалось, были получены впечатляющие результаты: удалось уменьшить блок данных размером 256 МБ до 0.5 кБ (0.0002%). При сжатии данных об участке вблизи поверхности земли, блок размером 256 МБ сократился до 5 МБ (2%), что так же дает существенную разницу: вместо одного участка мы теперь можем хранить одновременно 50, занимая тот же объем памяти.

Дальнейшая оптимизация описанного в статье алгоритмаимеет два ключевых направления. Во-первых, мы можем воспользоваться преимуществом независимости блоков данных внутри файла, для того чтобы реализовать параллельные алгоритмы компрессии, декомпрессии, чтения и записи. Во-вторых, линейная сложность алгоритма доступа к данным может быть сведена к логарифмической, путём введения более "умной" древовидной структуры, в рамках которой будет сделан возможным бинарный поиск.

Литература:

1. Лахов А.Я. Программное обеспечение для стереовизуализации результатов конечно-элементного моделирования[Электронный ресурс] / А.Я. Лахов// «Инженерный вестник Дона», 2013, №1 – Режим доступа: http://ivdon.ru/magazine/archive/n1y2013/1501 (доступ свободный) – Загл. с экрана. – Яз.рус.

2. Рачковская Г.С. Математическое моделирование и компьютерная визуализация сложных геометрических форм / Г.С. Рачковская// Инженерный вестник Дона, 2013, №1 – Режим доступа: http://ivdon.ru/magazine/archive/n1y2013/1498 (доступ свободный) – Загл. с экрана. – Яз.рус.

3. Чеканов Д. Рендеринг с помощью вокселей: новый уровень графики в играх?– Tom's Hardware URL: http://www.thg.ru/graphic/vox el_ray_casting/ one page.html Дата обращения20.09.2013

4. Arie Kaufman, DanielCohen,RoniYagel. Volume Graphics– IEEE Computer 1993, Vol. 26, №7, – pp. 51-64

5. Eric Lengyel. Voxel-Based Terrain for Real-Time Virtual Simulations. PhD diss., University of California at Davis, 2010, – pp. 13-21

6. SamuliLaine, TeroKarras.Efficient Sparse Voxel Octrees. – URL: http://www.nvidia.com/docs/IO/88889/laine2010i3d_paper.pdf Датаобращения 20.09.2013

7. Deflate compressed data format specification version 1.3. – URL: http://tools.ietf.org/html/rfc1951Датаобращения 20.09.2013

8. ZIP File Format Specification – URL: http://www.pkware.com/documents/ APPNOTE/APPNOTE-6.3.2.TXTДатаобращения 20.09.2013

9. Алгоритмысжатияизображений – URL: http://www.lib.ru/TECHBOOKS/ ALGO/VATOLIN/algcomp.htmДатаобращения20.09.2013

10. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2002. - 384 с.