УДК 004.4’413, 004.021 DOI: 10.17587/it.27.188-194

Д. В. Малаховецкий, студент, e-mail: dmitriymalax@yandex.ru,
Московский государственный технологический университет "СТАНКИН",
А. И. Разумовский, канд. техн. наук, ст. науч. сотр., e-mail: razumowsky@yandex.ru,
Институт проблем управления РАН, г. Москва

Синтаксический анализ символьных массивов методом рекурсивного охвата и структурирования на примере данных VRML-формата

Представлен новый подход к структуризации, сегментации и алгоритмическому оформлению синтаксического анализатора символьных массивов на примере данных VRML. Ключевой особенностью метода является возможность формирования иерархически-сложного объекта посредством рекурсивной структуризации данных, позволяющей охватывать в совокупности все содержимое объекта, включая его произвольную вложенность дочерних объектов. Это позволяет достичь высокой управляемости при разработке алгоритма синтаксического анализа и дает возможность сосредоточиться каждый раз на конкретном фрагменте данных, при этом не теряя из виду всей совокупной связности информации. Полученные результаты легко могут быть использованы при создании удобных структур хранения данных, связанных с обеспечением информационной безопасности, для решения проблемы сдерживания объема данных в файлах, в задачах управления большими данными в гетерогенных системах, а также при обработке иерархических данных в интернете вещей.

Ключевые слова: *метод анализа данных, синтаксический анализ, рекурсия, VRML-формат, полиморфная иерархия*

Введение

Всякий предметный или системный анализ необходим и востребован по причине содержательной сложности данных предмета или системы. Информационная сложность представляет собой нечто с трудом воображаемое либо не поддающееся мысленному охвату. Поэтому информацию перед непосредственным использованием необходимо соответствующим образом подготовить. Среди основных способов такой подготовки является анализ информации и ее практическая апробация. В современном мире обилие информации является нормой, поэтому востребованными остаются не только предоставляющие информацию ресурсы, но и методы, с помощью которых можно найти, соотнести, преобразовать нужную информацию, провести ее анализ и синтез.

В настоящем исследовании мы изучаем данные графического формата VRML (Virtual Reality Modeling Language), который был создан для моделирования виртуальной реальности. Формат, который набрал огромную популярность в 90-х годах и был вытеснен впоследствии своим преемником — языком моделирования виртуальной реальности X3D, по-прежнему вполне востребован. Область его применения простирается от интерактивной векторной графики и Web-технологий до приложений медицинской направленности, строительства, картографии и мультимедийных систем. Потребность в использовании этого формата по-прежнему высока в системах автоматизированного проектирования, включая обмен данными между различными графическими системами [1—4]. Язык оперирует объектами, описывающими геометрические фигуры и их расположение в пространстве [5].

Используя файлы символьного формата, язык VRML позволяет определить не только геометрические свойства объектов в трехмерном пространстве, например, расположение и форму сложных поверхностей и многогранников, но и физические данные об их цвете, текстуре, блеске, прозрачности, источниках освещения. Кроме того, в качестве реакции на действия пользователя или другие внешние события существует возможность дополнить описание модели, задав параметры движения, звуков, освещения и иных аспектов виртуального мира [6].

Прежде чем искать характерные алгоритмические пути синтаксического анализа данных, важно понимать, что анализ сообразуется с операциями последовательного считывания определенных порций данных из файла, поэтапной идентификации VRML-объектов по характерным признакам, сопоставления с уже имеющимися в хранилище данными и, наконец, формирования необходимой структуры для взаимодействия с данными объектами.

Как большинство языков, VRML имеет четкую конструкцию и определенный набор лексических, синтаксических и семантических правил, определяющих внешний вид програм-

мы и ее действия. Из этого можно сделать ложный вывод о том, что написание алгоритма синтаксического анализа не составит большого труда. Вся структура формата уже продумана разработчиками языка, и в таком случае необходимо только создать метод, который позволит ее правильно считать. Однако при написании программы возникают трудности, связанные с иерархическими особенностями языка.

Мы представляем новый подход к алгоритмической организации синтаксического анализа данных, который исключает проблемы, связанные с иерархической сложностью VRML информации. В основе метода лежат рекурсивное считывание объектов формата VRML, их последовательный анализ, идентификация и сопоставление с уже определенными данными. Для хранения содержательной структуры объектов используется специальный контейнер, который одновременно обеспечивает удобный доступ ко всему множеству объектов, а также возможность взаимодействия как с отдельными объектами, так и со всей структурой объектов в целом.

1. Методы исследования

Прежде чем выявить и сделать попытку разрешить проблемы синтаксического анализа сложноструктурированных данных, необходимо классифицировать виды их структурной организации.

В формате VRML имеются три вида описаний:

- простые объекты (без спецификации), которые последовательно считываются из файла и сразу же помещаются в соответствующее место полиморфной иерархии;
- "якорные" объекты (ЯО), выделяемые в формате ключевым словом "DEF". Они содержат определение и описание свойств объекта для последующего использования в иерархии. Порядок обработки таких объектов следующий: вначале считываются данные, имеющие полноту описания объекта, затем эти данные помещаются в отдельный контейнер DEF, которые хранит "якорные" объекты, идентифицируемые по ключу-имени. Далее указатель на такой объект помещается в соответствующее место иерархии;
- пользовательские объекты, определяемые спецификацией "USE": содержат указание на использование определенного "якорного" объекта, имеющегося в иерархии. В момент считывания пользовательского объекта выполняется поиск по соответствующему ключу в контейнере DEF, после чего найденный указатель размещается в хранилище.

Первая и самая очевидная проблема, с которой можно столкнуться при анализе VRML-данных, заключается в многократной вложенности объектов, которая в некоторых случаях является избыточной (рис. 1, см. третью сторону обложки).

Средствами языка C++ имеется возможность описать и реализовать вложенность, однако для этого объекты должны быть разных типов. В данных VRML нередко можно встретить, как один и тот же объект (например, объект типа Group) несколько раз наследуется сам от себя, что делает невозможным последовательное формирование структуры вложенных данных. На иллюстрации красной рамкой обозначена ситуация многократной вложенности объекта Group (рис. 1, см. третью сторону обложки).

Следующая проблема происходит из синтаксиса формата VRML. В языке VRML, как и во многих других языках, допускаются ЯО, которые могут быть описаны в файле только единожды, а используются затем неопределенное число раз посредством именной ссылки. Иными словами, это определения комплексных типов данных с возможностью созидания сложно сгруппированных объектов. Затруднения здесь заключаются в вероятной потере контроля над динамической памятью, которая выделяется всякий раз при считывании данных. Так как ЯО может использоваться в алгоритме не один раз, необходимо определенно знать, в какой именно момент его следует из памяти удалять. Иначе говоря, за ЯО следует установить дополнительное наблюдение.

С последней проблемой можно столкнуться не всегда, а только при считывании многофайлового объекта — VRML-сборки. Синтаксический анализ и считывание сборок организуется аналогично однофайловым данным. Однако казус заключается в том, что один и тот же файл может использоваться в сборке многократно. Это означает, что однозначная идентификация объектов в такой сборке затруднена. Кроме того, возникает утечка памяти из-за

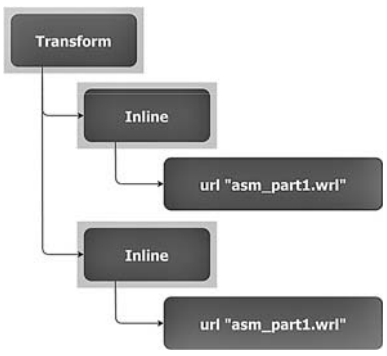

Рис. 2. Пример повторного использования файла в сборке

перезаписи указателей на ЯО, которые, имея одно и тоже имя, в разных файлах естественно ведут себя по-разному (рис. 2).

Память под такие объекты выделяется в общем порядке, однако она может быть вовремя не освобождена, так как указатели на них перезаписываются в ходе последующего считывания данных, ссылающихся на тот же самый объект.

1.1. Создание универсального контейнера VRML-объектов

Для того чтобы разрешить проблему вложенности объектов при создании полноценной структуры VRML, важно также понять семантику VRML-формата. Так, имеется определенный объект-родитель, в котором содержатся какие-либо свойства, и массив дочерних элементов (children). Такой массив включает, в свою очередь, множество подобных объектов, у которых имеются индивидуальные свойства и собственный массив наследуемых объектов. При этом такое представление по-прежнему не дает ответа на главный вопрос: как справиться с проблемой многократного наследования? Ведь массив дочерних объектов (children) может содержать практически любой произвольный объект формата VRML: Transform, Group, Shape и так далее. Мы наблюдаем здесь, как сложность охвата данных только возрастает. Необходим нетривиальный взгляд на многократно вложенные данные. Иначе говоря, требуется выразить вложенность абстрактно, сохраняя при этом индивидуальные черты формата данных.

Проведем классификацию типов. Вся структура VRML состоит из так называемых узлов (Node) [5,7], которые подразделяются на геометрические узлы (Box, Cone, Cylinder, IndexFaceSet) и группирующие узлы (Group, Transform, Collision). Кроме того, есть узлы, которые отвечают за графическое представление (Appearance), а также узлы, определяющие свойства геометрических объектов (geometry). Соответственно, ориентируясь на такую классификацию, несложно сформировать полиморфную иерархию на языке C++.

Определим абстрактный класс Node как обобщение всевозможных узлов формата VRML. Затем установим наследование прочих классов от абстрактного класса Node. Таким образом, мы получаем системную связность произвольных элементов данных посредством включения их типов в общую полиморфную иерархическую композицию.

Далее, поскольку любой тип, образованный на основании перечня ключевых слов формата VRML, может быть включен в иерархию, появляется возможность создать массив указателей на объект базового типа данных (Node*) этой иерархии. Объект любого типа, входящего в иерархию, может быть помещен в этот массив в качестве указателя на динамически выделенную под объект память. Таким способом получится легко осуществить реализацию хранилища VRML-данных, сохраняя при этом их разнообразие.

1.2. Косвенный мониторинг "якорных" объектов

Для хранения ЯО и последующего их использования при построении полиморфной иерархии выбран ассоциативный контейнер стандартной библиотеки шаблонов std::map. Ключом в нем устанавливается имя считываемого объекта, а значением — указатель на динамический ЯО.

В процессе считывания данных выполняется проверка — является ли объект "якорным"? В положительном случае его указатель помещается в глобальный контейнер DEF по ключу — имени ЯО. Впоследствии при считывании пользовательского объекта выполняется поиск по ключу в контейнере DEF, после чего под него выделяется память, а сам объект помещается в иерархию данных. Если же объект не является "якорным", то под него сразу выделяется память, и он поступает в целевое хранилище.

Главным хранилищем данных является динамический массив nodes (узлы) стандартной библиотеки шаблонов, типа std::vector<Node*>. Он предназначен для сбора всей полиморфной иерархии считываемых объектов. Провести идентификацию — является ли текущий объект "якорным", можно посредством проверки именной строки def. Если она отсутствует у объекта, тогда объект просто помещается в главное хранилище. В противном случае объект квалифицируется как ЯО, он также помещается в хранилище std::map<string, Node*> DEF в целях косвенного мониторинга его последующего использования:

```
if (group->def! = "") {
DEF->insert(std::pair<string, Node*>(def, group));
def.clear();
}
```

Если при считывании обнаруживается пользовательский объект ("USE"), тогда по его имени выполняется поиск в хранилище DEF по ключу (имя объекта + имя файла), и найденный там объект помещается в главный контейнер nodes:

```
if (str = = "USE") {
file >> str;
shape->geometry = (Geometry*)DEF->at(str + "_"
+ cur_file);
}
```

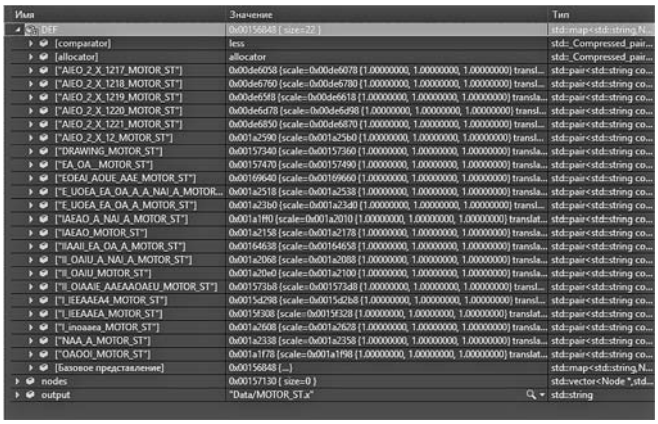

Рис. 3. Содержимое хранилища ЯО

Таким образом, все ЯО хранятся в контейнере DEF (рис. 3).

В главном хранилище, которое обеспечивает содержание иерархической структурной связности объектов, содержатся указатели на ЯО из DEF, а также объекты без специализации. Для того чтобы корректно выполнить удаление и очистку всей структуры данных, необходимо перед удалением объекта выполнять проверку — является ли он "якорным"? Если — нет, то удаление должно выполняться сразу, иначе удаление проводится по завершении всего синтаксического анализа.

1.3. Идентификация ЯО

Для осуществления однозначной идентификации по имени объекта необходимо образовывать комплексное именование, добавляя к имени некоторый суффикс. Такая дополнительная именная добавка позволит различать ЯО при считывании VRML-сборок, в которых названия объектов-переменных могут быть одинаковыми. Таким суффиксом удобно определить название файла, в котором находится считываемый объект.

Однако проблема с неоднозначной идентификацией переменных при использовании одного файла несколько раз при этом все же остается, поскольку имена файлов и имена объектов могут совпадать. Для решения данной проблемы необходимо создать дополнительное множество соотносимых через имя элементов. В стандартной библиотеке шаблонов существует весьма удобный для достижения этой цели ассоциативный контейнер std::map:

std::map<string, vector<Node*>*>* FILES

В объекте FILES ключом будет выступать название файла, считанного ранее, а значением — указатель на соответствующим образом обработанный иерархический объект. При считывании имени файла выполняется поиск по контейнеру FILES. И если ранее это файл не был проанализирован, то проводится последовательное чтение объектов из этого файла с созданием соответствующей иерархии объектов, указатель на которую используется для добавления в хранилище FILES.

Данный способ идентификации переменных имеет ряд преимуществ:

- однозначная идентификация ЯО при считывании сборки с одинаковыми файлами;
- однозначная идентификация ЯО при считывании сборки с различными файлами, в которых названия переменных при этом являются одинаковыми;
- оптимизация времени выполнения программы за счет пропуска повторной процедуры считывания уже имеющихся данных.

1.4. Метод рекурсивного охвата данных

По мере выявления проблем и прояснения их особенностей важно правильно подобрать средства и способы достижения целевого результата. Итак, необходимо реализовать совокупность методов, которые позволят следующее:

- провести чтение данных из файла VRML-формата;
- выполнить разбор на лексемы считанных данных;
- идентифицировать лексемы в объекты и сопоставить их для установления связей между ними и прочих свойств;
- построить полиморфную иерархию на основе полученных объектов.

Проблема необходимости дополнительного мониторинга ЯО, а также неопределенности уровня и степени вложенности объектов на этапе чтения данных приводит к необходимости идентифицировать объекты так, словно уже известна их глубина вложенности и свойства. Для этого удобно использовать рекурсивный подход, который позволит охватить данные в их предположительной целостности.

Основной процедурой, которая осуществляет чтение формата VRML, установим функцию *parseNode* (рис. 4).

Настоящая функция принимает следующие параметры:

- объект файлового потока, откуда должно выполняется чтение данных;
- контейнер, в который должны быть помещены все считанные объекты;
- строка, определяющая текущее положение указателя на данные в файле;
- имя текущего файла для создания уникальных имен объектов.

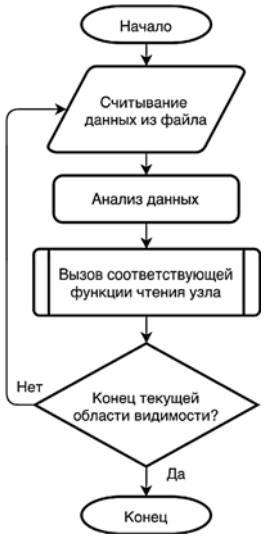

Рис. 4. Блок-схема алгоритма чтения VRML-файла parseNode

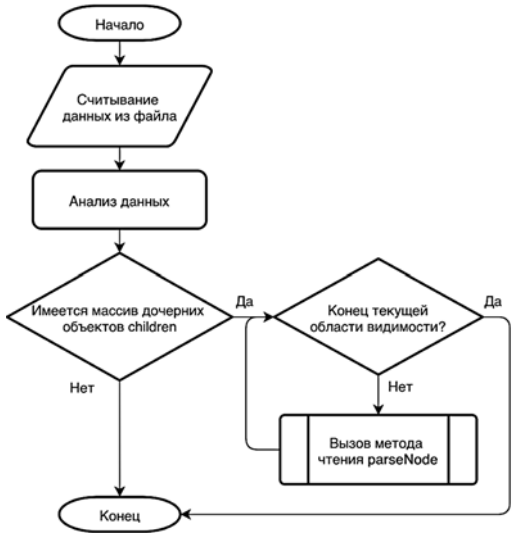

Рис. 5. Блок-схема алгоритма чтения группирующих узлов parseGroup

Рис. 6. Блок-схема алгоритма считывания VRML-сборок

В зависимости от квалификации объекта — Transform, Group, Shape, Inline — осуществляется вызов соответствующей функции для установления его свойств. Кроме того, выполняется запись наименования объекта для последующей ассоциации и идентификации ЯО. К имени объекта также добавляется имя текущего файла, из которого он был считан.

Реализация всех методов чтения группирующих узлов — Group, Transform, Collision и прочих — наиболее наглядно видна на примере алгоритма рекурсивного чтения объекта Group — *parseGroup*, так как группирующие узлы отличаются только своими свойствами, и важно сосредоточиться на реализации чтения их дочерних объектов (рис. 5).

В случае, если объект имеет массив зависимостей children, организуется цикл, в котором итерационно вызывается метод *parseNode* и проводится рекурсивное чтение дочернего объекта с размещением затем в контейнере children.

Функция *parseNode* имеет следующие входные параметры:

- объект текущего файлового потока;
- строка, которая определяет позицию указателя в файле;
- название настоящего файла для создания уникальных имен объектов;
- контейнер, в который должны быть помещены считанные объекты. Здесь в качестве контейнера передается контейнер children текущего объекта.

Аналогично рекурсивно охватывается вся иерархическая цепочка дочерних объектов: если дочерний объект также является группирующим узлом, то при его чтении запускается аналогичный цикл.

Решение проблемы однозначной идентификации объектов при чтении сборочного VRML-файла отведено функции O*riginality*. Эта операция отвечает за непосредственное наращивание полиморфной иерархии. Основной чертой функции является проведение однозначной идентификации объектов с применением контейнера FILES, описанного выше (рис. 6).

Реализация метода *Originality* делится на два этапа:

1. Считывание названия файла.
2. Проверка того факта, был файл считан ранее или нет.

2. Полученные результаты

По мере чтения VRML-файла (или сборки) программа рекурсивно выстраивает полиморфную иерархию всех объектов, их свойств и наследников (рис. 7).

Результирующая иерархия представлена в древовидном виде и хранится в объекте типа std::vector. Такой способ хранения позволяет наилучшим образом осуществлять удобный доступ ко всем объектам, а также простоту взаимодействия с ними. Кроме того, посредством метода рекурсивного охвата достаточно легко проводить дополнительный мониторинг ЯО, поскольку каждый ЯО наблюдается одновременно и как отдельный объект, и как содержащий множество, объем которого заранее неизвестен, прочих объектов.

Формат представления VRML-структуры в качестве полиморфной иерархии имеет ряд следующих преимуществ:

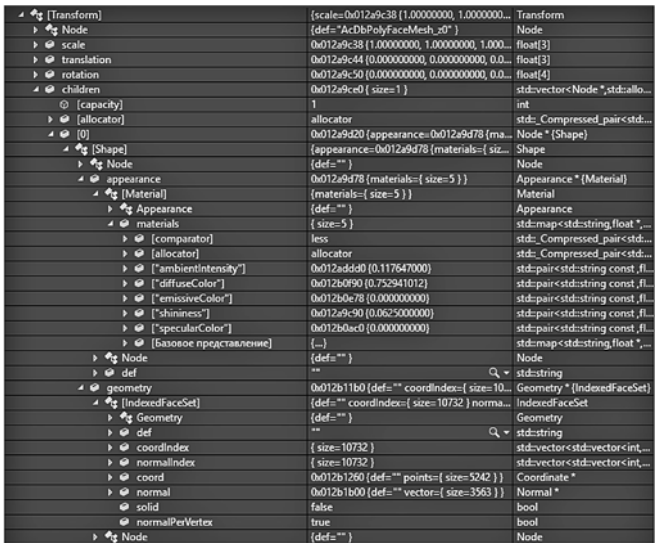

Рис. 7. Фрагмент содержимого хранилища VRML-данных nodes

- возможность оперировать отдельными объектами в контексте всего их иерархического устройства;
- возможность обрабатывать весь массив накопленных данных единообразным способом, поскольку взаимодействие с каждым объектом использует функционал полиморфизма объектно-ориентированного подхода.

Последнее преимущество способствует более эффективному способу вывода всех накопленных данных в файл формата иной конфигурации.

Заключение

Всякая информация имеет индивидуальную сложность, которая начинает проявляться при анализе и разборе данных. Первоначальное предположение о простоте синтаксического анализа в связи с тем, что VRML видится четко структурированным языком, оказывается обманчивым. При более пристальном взгляде на задачу выявляются проблемы, связанные с поиском верного пути преодоления барьеров сложности.

В выборе метода синтаксического анализа важно обращать внимание на детали связности элементов информации, а также на последовательность их обработки и преобразований.

Метод рекурсивного охвата данных, представленный в статье, предлагает простое и элегантное решение проблем информационной связности и порядка обработки данных. Посредством рекурсивного способа чтения и структуризации объектов оказывается возможным создавать массивные иерархические структуры, с элементами которых впоследствии можно легко взаимодействовать. Это открывает возможность оперировать объектами не только как отдельными единицами, но и с учетом всей иерархической структуры.

Представленный метод синтаксического анализа данных дает возможность, последовательно считывая данные, формировать "на лету" иерархию объектов непосредственно на основе структуры их формата. Иначе говоря, прямая выгода этого подхода заключается в отсутствии дополнительных преобразований и структуризации при считывании данных из файла. Это сокращает время считывания, а также понижает сложность программного кода.

Наиболее важный вывод состоит в том, что благодаря методу рекурсивного охвата данных оказывается возможным программно воспринимать и преобразовывать любые иерархические массивы символьных данных. Предложенный подход можно расширить в плане исследований параллельной обработки данных, а также для создания удобных структур хранения данных, связанных с обеспечением информационной безопасности [8]. Актуальными остаются проблемы сдерживания объема данных в файлах и повышения скорости считывания [9] — здесь также есть широкое поле приложений представленного метода. В плане управления большими данными [10, 11], в связи с появлением гетерогенных и гибридных систем хранения, таких как Hadoop и Spark, видится значительная перспектива применения данного метода. Наконец, широкий горизонт открывается в создании и обработке иерархических данных, используемых в интернете вещей (IoT) [12], что сулит значительные выгоды в восприятии распределенной информации.

Список литературы

1. **Ziwar F., Elias R.** VRML to WebGL Web-based converter application // 2014 International Conference on Engineering and Technology (ICET). IEEE. 2014, April. P. 1—6.

2. **Whyte J., Bouchlaghem N., Thorpe A., McCaffer R.** From CAD to virtual reality: modelling approaches, data exchange and interactive 3D building design tools // Automation in construction. 2000. Vol. 10, N. 1. P. 43—55.

3. **Pazlar T., Turk Ž.** Interoperability in practice: geometric data exchance using the IFC standard // Journal of Information Technology in Construction (ITcon). 2008. Vol. 13, N. 24. P. 362—380.

4. **Huber D.** The ASTM E57 file format for 3D imaging data exchange // Three-Dimensional Imaging, Interaction, and Measurement. International Society for Optics and Photonics.2011, January. Vol. 7864. P. 78640A.

5. **Taubin G., Horn W. P., Lazarus F., Rossignac J.** Geometry coding and VRML // Proceedings of the IEEE. 1998. Vol. 86, N. 6. P. 1228—1243.

6. **Nadeau D. R.** Building virtual worlds with VRML // IEEE Computer Graphics and Applications. 1999. Vol.19, N. 2. P. 18—29.

7. **Pesce M.** VRML browsing and building cyberspace (No. 006 P473). New Riders Publishing, 1995.

8. **Tamassia R., Triandopoulos N.** Computational bounds on hierarchical data processing with applications to information security // International Colloquium on Automata, Languages, and Programming. Springer, Berlin, Heidelberg. 2005, July. P. 153—165.

9. **Schulze S. F., LaCour P., Buck P. D.** GDS-based mask data preparation flow: data volume containment by hierarchical data processing // 22nd Annual BACUS Symposium on Photomask Technology. International Society for Optics and Photonics. 2002, December. Vol. 4889. P. 104—114.

10. **Krish K. R., Wadhwa B., Iqbal M. S., Rafique M. M., Butt A. R.** On efficient hierarchical storage for big data processing // 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE. 2016, May. P. 403—408.

11. **Wang X., Yang L. T., Kuang L., Liu,X., Zhang Q., Deen M. J.** A tensor-based big-data-driven routing recommendation approach for heterogeneous networks // IEEE Network. 2019. Vol. 33, N. 1. P. 64—69.

12. **Wan S., Zhao Y., Wang T., Gu Z., Abbasi Q. H., Choo K. K. R.** Multi-dimensional data indexing and range query processing via Voronoi diagram for internet of things // Future Generation Computer Systems. 2019. Vol. 91. P. 382—391.

D. V. Malakhovetsky, Student, e-mail: dmitriymalax@yandex.ru,
Moscow State University of Technology "STANKIN", Moscow, Russian Federation,
A. I. Razumowsky, PhD in Technical Sciences, Senior Researcher, e-mail: razumowsky@yandex.ru,
Trapeznikov Institute of Control Science RAS, Moscow, Russian Federation

Parsing Character Arrays by Recursive Scoping and Structuring Using the Example of VRML Data

Parsing character arrays by recursive scoping and structuring using the example of VRML data. The article presents a new method for structuring, segmentation and algorithmic design of the parser of character arrays using the example of VRML data. The key feature of the method is the ability to form a hierarchically complex object by means of recursive data structuring, which makes it possible to cover in aggregate the entire contents of the object, including its arbitrary nesting of child objects. This leads to high controllability of the development of the parsing algorithm, allowing you to focus each time on a specific piece of data, while not losing sight of the entire aggregate connectivity of information. The results obtained can easily be used in plans for creating convenient data storage structures related to information security, solving the problem of containing the amount of data in files, managing big data in heterogeneous systems, and processing hierarchical data in the Internet of Things.

Keywords: *data analysis method, parsing, structuring, segmentation, recursion, VRML-format, polymorphic hierarchy*

DOI: 10.17587/it.27.188-194

References

1. **Ziwar F., Elias R.** VRML to WebGL Web-based converter application, *2014 International Conference on Engineering and Technology (ICET)*, IEEE, 2014, April, pp. 1—6.

2. **Whyte J., Bouchlaghem N., Thorpe A., McCaffer R.** From CAD to virtual reality: modelling approaches, data exchange and interactive 3D building design tools, *Automation in Construction*, 2000, vol. 10, no. 1, pp. 43—55.

3. **Pazlar T., Turk Ž.** Interoperability in practice: geometric data exchance using the IFC standard, *Journal of Information Technology in Construction (ITcon),* 2008, vol. 13, no. 24, pp. 362—380.

4. **Huber D.** The ASTM E57 file format for 3D imaging data exchange, *Three-Dimensional Imaging, Interaction, and Measurement,* International Society for Optics and Photonics, 2011, January, vol. 7864, pp. 78640A.

5. **Taubin G., Horn W. P., Lazarus F., Rossignac J.** Geometry coding and VRML, *Proceedings of the IEEE*, 1998, vol. 86, no. 6, pp. 1228—1243.

6. **Nadeau D. R.** Building virtual worlds with VRML, *IEEE Computer Graphics and Applications,* 1999, vol.19, no. 2, pp. 18—29.

7. **Pesce M.** VRML browsing and building cyberspace (No. 006 P473), New Riders Publishing, 1995.

8. **Tamassia R., Triandopoulos N.** Computational bounds on hierarchical data processing with applications to information security, *International Colloquium on Automata, Languages, and Programming,* Springer, Berlin, Heidelberg, 2005, July, pp. 153—165.

9. **Schulze S. F., LaCour P., Buck P. D.** GDS-based mask data preparation flow: data volume containment by hierarchical data processing, *22nd Annual BACUS Symposium on Photomask Technology*, International Society for Optics and Photonics, 2002, December, vol. 4889, pp. 104—114.

10. **Krish K. R., Wadhwa B., Iqbal M. S., Rafique M. M., Butt A. R.** On efficient hierarchical storage for big data processing, *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid),* IEEE, 2016, May, pp. 403—408.

11. **Wang X., Yang L. T., Kuang L., Liu,X., Zhang Q., Deen M. J.** A tensor-based big-data-driven routing recommendation approach for heterogeneous networks, *IEEE Network*, 2019, vol. 33, no. 1, pp. 64—69.

12. **Wan S., Zhao Y., Wang T., Gu Z., Abbasi Q. H., Choo K. K. R.** Multi-dimensional data indexing and range query processing via Voronoi diagram for internet of things, *Future Generation Computer Systems*, 2019, vol. 91, pp. 382—391.