Объектно-ориентированный подход к построению интерпретатора языка Adobe PostScript.

А.Г.Соловьев, М.В.Шишалов
Нижегородский государственный университет,
факультет ВМК

Н.Новгород, Россия

АННОТАЦИЯ

Поводом к выполнению описываемой ниже разработки явилось создание серии прецизионных лазерных фотоплоттеров КОРАЛ, сконструированных и выпускаемых малой серией в научно-исследовательском институте прикладной математики и кибернетики (НИИ ПМК Нижегородского государственного университета). Драйверы этих устройств ориентированы на простейший штриховой формат. Поэтому для расширения сферы их применения приходится разрабатывать различные конверторы с наиболее распространенных графических форматов. К моменту начала нашей работы плоттеры КОРАЛ "научились понимать" растровый формат BMP и векторный формат HP-GL, активно используемые в отечественных автоматизированных картографических системах. Однако в области полиграфии наибольшим авторитетом пользуется промышленный стандарт **Adobe PostScript.** В предлагаемой работе описывается подход к созданию интерпретатора этого языка и обсуждаются наиболее важные решения, ориентированные на повышение прозрачности и эффективности программного продукта.

1. ТЕХНОЛОГИЯ ADOBE POSTSCRIPT.

Одним из наиболее дорогостоящих программных продуктов электронной полиграфии являются так называемые **RIP**'ы **(Raster Image Processing)** – программы, обеспечивающие вывод текстовых документов и графических изображений на различные фотонаборные автоматы с высокой разрешающей способностью.

Существенный вклад в подготовку электронных изданий, не зависящих от платформы, внесла небольшая американская фирма Adobe System Incorporated, основанная в 1982 г. Главным ее изобретением явился язык **PostScript**, на базе которого был развит универсальный формат **PDF (Portable Documents Format)** и целая серия программных пакетов Adobe Acrobat, Adobe PostScript и др.

Язык **PostScript** – простой интерпретирующий язык программирования с мощными графическими возможностями. Его первичной задачей является описание в текстовом виде графических объектов и простых изображений на странице, согласно модели отображения фирмы **Adobe**. Описание страницы является описанием высокого уровня и не зависит от устройства. Описание страницы на **PostScript** может содержать следующие элементы:

- описание произвольных фигур, созданных из отрезков, дуг, прямоугольников и кубических кривых.
- графические операторы, которые позволяют при выводе фигуры использовать линии любой толщины, заполнение любым цветом и т.д. Цвета могут быть определены различными способами: полутонами, схемами **RGB**, **CMYK** и др.
- возможности работы с глифами шрифта, как с графическими фигурами, к которым можно применять любые обычные графические операции, например, заливка, штриховка, пересечение, объединения и т.д.
- простые растровые изображения. Язык **PostScript** может содержать описания растровых изображений с любым разрешением и цветовой моделью.
- описание аффинных преобразований системы координат, включая сдвиг, масштабирование, поворот, отражение и растяжение/сжатие. Эти преобразования применяются одинаково ко всем элементам страницы, включая текст, графические объекты и простые изображения.

Язык **PostScript** может рассматриваться как

- язык программирования обычного назначения с мощной встроенной графикой.
- диалоговая система управления растровыми устройствами
- язык описания страниц
- программно и устройство независимый формат описания страниц

Язык описаний страниц может иметь статический или динамический формат.

Статический обеспечивает определенный встроенный набор действий и синтаксиса; классический пример – коды управления форматом для принтера. Исторически статические форматы были разработаны для определенного класса устройств.

Динамический имеет бóльшую гибкость и может быть расширен. Операторы языка можно перекрывать в случае необходимости, изменяя поведение программы.

PostScript – язык описания страниц динамического формата.

Обычно прикладные программы типа систем составления документов, иллюстраторов и систем машинного проектирования генерируют описания

страницы на **PostScript** автоматически. Программисты используют **PostScript** только для того, чтобы воспользоваться специальными возможностями языка или в целях отладки интерпретатора.
Хорошо структурированное описание **PostScript** состоит из двух частей: **пролог**, сопровождаемый **сценарием**. Нет ничего в языке, что формально отличает пролог от сценария, это просто соглашение, но весьма полезное. Его рекомендуется поддерживать.
Пролог – набор описаний процедур, которые могут использоваться при выполнении сценария. Главной задачей пролога является описание основного словаря, используемого при интерпретации сценария.
Сценарий – создается для описания отдельных элементов страницы. Он состоит из операторов **PostScript** и процедур пролога вместе с операндами и данными.
Деление **PostScript-**программы на пролог и сценарий уменьшает размер описания страницы. Вот, например, одна из причин такого разделения: одной из распространенных задач является размещение текста в некоторой позиции; это действие состоит из двух: перемещение текущей точки и рисование текста. Программа, часто использующая эту операцию, определит в прологе процедуру, комбинирующую эти действия:

```
/ms { moveto show } bind def
```

Позже, в сценарии процедура может быть вызвана с конкретными данными:

```
(sometext) 100 200 ms
```

Часть сценария обычно содержит последовательность отдельных страниц. Каждая страница должна основываться только на определениях своего пролога, а не на установках предыдущей страницы сценария. Язык включает средства обслуживания, гарантирующие независимость страниц.

На данный момент выпущено уже три версии языка (**Language Level**). Каждая новая версия не изменяет принципов работы функций предыдущей версии, а лишь добавляет новые операторы и параметры, оптимизируя работу и расширяя возможности.
В уровень 2 добавлено явное построение словарей (синтаксис похож на описание массива), выбор цветов в пространстве **CMYK**, поддержка **пользовательских путей** (сохранение путей для последующего использования в составе других путей или для повторного использования), поддержка **полутонов** и произвольных цветовых пространств. Для закраски можно применять кисти на основе **образцов** (**Pattern**) – как правило, растровых. При работе с файлами можно использовать **фильтры** (аналогично препроцессору в **C** они дополнительно переводят программу в некоторый иной вид перед интерпретацией). Интерпретаторы уровня 2 получили возможность работы с **упакованными массивами** и бинарным кодированием объектов. Кроме того, введен **кэш путей** для ускорения работы графики и поддержка устройств, допускающих печать поверх уже напечатанного.

Новые возможности уровня 3 следующие:
1) сохранение / восстановление текущего **пути отсечения**,
2) **контроль точности** сглаживания при отображении (**smoothness**).
3) **градиентные заливки** фигур и использование **словарей градиентных заливок**, которые могут быть радиальными, линейными, типа Гуро и произвольно задаваемыми с помощью математических функций.
4) Поддержка интерпретаторами **распознавания идиом** (возможность при интерпретации заменить часть набора команд на другой набор, который делает то же самое, но более эффективно за счет использования новых средств языка).
5) Кодирование/декодирование с помощью **LZW**.

Словари.
Словарем называется **ассоциированная таблица**, элементы которой – пары объектов **PostScript**. Первый элемент называется ключом, второй – значением. **PostScript** включает операторы вставки в словарь, поиска ключа, получения ассоциированного значения и его исполнения. Ключи обычно являются именами, однако, ключом может быть любой объект, кроме **null**. Если попытаться использовать строку в качестве ключа, интерпретатор сначала переведет ее в имя.
Имена и строки взаимозаменяемы, когда используются в качестве ключей в словарях.
При создании словаря указывается его вместимость. При переполнении словаря в уровне 1 выдается ошибка, в уровне 2 и 3 автоматически увеличивается вместимость словаря. Словари обычно связывают имена и значения компонентов программ, таких как переменные, процедуры, аналогично использованию идентификаторов в других языках. Но есть и другие применения: например, программа может содержать словарь, связывающий имена символов с процедурой их рисования.
Есть 3 способа работы со словарями:
- операторы доступа к определенному словарю как к операнду
- текущий словарь и набор операторов неявного доступа к нему
- явный поиск исполнимых имен, встречающихся в программе.

Интерпретатор поддерживает стек словарей; словари могут быть помещены (**push**) и извлечены (**pop**) из стека. Словарь на вершине стека называется **текущим**.
Когда интерпретатор ищет ключ неявно, например, исполняет имя объекта, сначала анализируется текущий словарь, потом более низкие, пока ключ не будет найден или не исчерпается стек.
В уровне 1 имеется два встроенных словаря: **systemdict** и **userdict**, в уровнях 2 и 3 – три словаря: **systemdict, globaldict** и **userdict**.
- **Systemdict** – словарь «**ReadOnly**», связывающий все операторы **PostScript** с их значениями (действиями). Содержит все определения, включая стандартный локальный и глобальный словари, а

также поименованные константы, такие как **true** и **false**.

- **Globaldict** – словарь с возможностью записи, находящийся в глобальной **VM**.
- **Userdict** - находящийся на вершине стека словарей. Оператор **def** помещает здесь определения функций и переменных. Приложения могут создавать свои словари или использовать пользовательский словарь.

Анализируя пролог, программа заполняет **userdict** определениями функций, которые не входят в стандартный набор, но чаще, чем стандартные, используются в данной программе. Это позволяет сократить объем программы.

2. ИНТЕРПРЕТАТОР.

Язык **PostScript** является языком интерпретирующего типа. Исполнение программы состоит в последовательном выполнении потока команд. В языке **PostScript** понятие «исполнение» распространяется и на данные. Исполнение операторов – это исполнение встроенных действий, исполнение данных – это помещение их в стек выполнения.

Интерпретатор реализован на **Borland C++ Builder Enterprise Edition 5.0.**

3. СТРУКТУРЫ ДАННЫХ ИНТЕРПРЕТАТОРА.

Объекты **PostScript** реализованы классами. Общим предком всех классов был выбран класс **TNull**, соответствующий в **PostScript** пустому объекту. В нем собраны общие для всех классов свойства и виртуальные методы.

Поскольку объекты **PostScript** разделены на 2 большие группы (простые и составные), дерево наследования объектов имеет 2 поддерева. Общим предком для всех простых объектов был выбран объект **TInteger**, соответствующий в **PostScript** целым числам, а для составных – **TArray** (соответствующий массивам).

При реализации таких методов, как распознавание объекта из строки, преобразование его в строку, определение **"runtime"** имени и типа, применен полиморфизм.

4. ОБЩИЕ ПРИНЦИПЫ ИНТЕРПРЕТАЦИИ.

Интерпретатор считывает из файла поток команд, выделяет синтаксические конструкции и производит их выполнение.

Выделением синтаксических конструкций занимается **сканер**. Синтаксические конструкции отделяются друг от друга «белыми символами» (к ним относятся «пустой символ», TAB, «перевод строки», «возврат каретки», пробел и др.). Исключение составляет строка. Задание строки в **PostScript** начинается с открывающей круглой скобки, заканчивается закрывающей круглой скобкой, между которыми могут быть любые символы, включая «белые», например,

(Hello, world ! ! !) описывает строку " Hello, world ! ! ! "

По выделенной синтаксической конструкции интерпретатор создает объект требуемого типа с соответствующими атрибутами выполнимости и доступа. Значение объекта распознается из строки, описывающей синтаксическую конструкцию.

Для представления синтаксических конструкций в интерпретаторе выбран тип «массив **char**», так как первоначальная реализация, использующая тип **AnsiString**, показывала неудовлетворительные результаты в плане быстродействия. Например, файл размера 181 кБ интерпретировался на процессоре типа Pentium-3 за 8,2 с, что соответствует скорости 22,07 кБ/с. Замена типов данных **AnsiString** на **char*** позволила повысить скорость обработки этого же файла почти в 16 раз (362 кБ/с).

При создании объекта типичный путь – динамическое выделение памяти под новый экземпляр класса. Так как многие объекты **PostScript** короткоживущие, происходит создание/удаление порядка 10,000 объектов на файл размера порядка 1 МБ. Если отказаться от операций динамического выделения памяти в процессе интерпретации, то расчетное уменьшение времени работы должно составить $_{1,5\cdot10^{-4}}$ с на объект, т.е. 1,5 с на 1МБ, что в сравнении с предыдущим значением скорости дает увеличение до 787 кБ/с.

Для реализации этого был написан аллокатор, который до запуска программы на интерпретацию резервирует некоторое количество указателей под объекты часто используемых типов и выделяет под них память. В процессе интерпретации, когда требуется создать объект, аллокатор возвращает свободный указатель нужного типа и помечает его как «занятый». При удалении объекта, если указатель зарезервированный, происходит всего лишь пометка его как «свободного». В случае, когда все указатели оказываются занятыми, выделение памяти происходит как обычно.

Анализируя необходимые размеры массивов резервированных указателей на различных **PostScript** -программах, мы выяснили, что этот размер мало зависит от размера **PostScript** -программы и определяется только размерами пролога и пользовательских функций. В последней реализации размер массивов резервированных указателей установлен равным 40. Если сделать этот размер слишком большим, затраты на поиск свободного указателя превысят затраты на выделение памяти и использование аллокатора станет неэффективным.

При создании объекта требуется определить его тип и преобразовать в зависимости от типа строку синтаксической конструкции либо в целое число, либо в действительное число, либо в строку, либо в какой-нибудь тип. Чаще всего в **PostScript** -программах встречаются действительные числа, поэтому еще один способ ускорить интерпретацию – это ускорить перевод строк в **double** (тип языка **C**). Была написана функция, использующая массив **double**, хранящий числа вида $N*10^k$, где N, k – индексы массива. Массив заполняется до интерпретации. Во время перевода строки в **double** исполь-

зуется только операция сложения **double**, которая занимает 12 наносекунд на **Pentium** 100. Ускорение по сравнению со стандартной функцией **C** перевода строки в **double** составило порядка 8 раз.

PostScript различает непосредственное и отложенное выполнение. Большую часть работы происходит непосредственное выполнение, при котором объекты типов данных помещаются в стек данных, а для объектов исполнимого типа выполняется связанное с ними действие.

При исполнении объекта, имеющего атрибут «исполняемый» производится поиск в стеке словарей, начиная с вершины. Как правило, там находится **userdict**, в котором хранятся пользовательские функции. Если пользователь создает словари сам, то они располагаются выше **userdict**. При успешном нахождении для объекта вызывается ассоциированное действие. Если поиск не удался, то ищем в словаре, который находится глубже текущего. На самом дне стека расположен **systemdict**. В словарях выше **systemdict** ассоциированное значение описано операторами **PostScript**, а в **systemdict** описываются встроенные операторы **PostScript**, для них действие – это функция, реализованная на **C++**. Связь с операторами осуществляется с помощью **Стандартного кодирования системных имен PostScript**. В интерпретаторе это представлено ассоциированной таблицей, связывающей имена и коды. Если отказаться от поиска **в userdict** и сначала искать в **systemdict**, то, как правило, интерпретация происходит быстрее, но мы не реализовали этот подход, поскольку это противоречит принципам языка **PostScript**. Вообще нашей целью являлось написание не как можно более быстрого интерпретатора, а программы, которая работала бы с **PostScript-**файлами в соответствии с технологией, разработанной фирмой **Adobe**.

Реализация отдельных функций, входящих в стандарт **PostScript** производилась на основе шаблонов. Это было сделано главным образом, чтобы предотвратить утечки памяти из-за несбалансированности создания и удаления объектов. Каждая функция должна обратившись к стеку, прочитать нужное ей количество аргументов. Ответственность за распознавание типа и доступ лежит целиком на функции. После создания объекта, представляющего результат, функция должна положить его в стек и освободить память, занимаемую прочитанными из стека аргументами. Освобождать память, занимаемую результатом, в целях избежания ошибок работы с памятью, нельзя. Эти обязательные части каждой функции и составили шаблон.

5. ГРАФИКА POSTSCRIPT И ЕЕ РЕАЛИЗАЦИЯ В ИНТЕРПРЕТАТОРЕ.

Центральными понятиями графики языка **PostScript** являются **путь** (**path**) и **область** (**region**). Пути позволяют описывать линии на плоскости. На них накладывается очень мало ограничений. Пути могут быть как простыми, например, контур многоугольника, так и сложными, когда путь состоит из нескольких не связанных друг с другом контуров или имеет самопересечения, петли и другие сложные особенности. Несмотря на все многообразие кривых, **PostScript** строит пути с помощью только двух видов элементов: это отрезок прямой и кривая, которая носит название «кривая Безье», или «кубический сплайн». Кривые Безье были описаны в 60-х годах XX века и нашли широкое применение в системах автоматического проектирования, а также в дизайне и издательском деле, благодаря удобству их использования. Задать кривую Безье можно 4 точками, в которых две являются конечными (через них проходит кривая), а две – контрольными (с их помощью определяются вектора касательных к кривой в конечных точках). При аффинном преобразовании кривая Безье переходит в кривую Безье, и достаточно перевести аффинно 4 точки и построить кривую в новых координатах по тем же алгоритмам.

С помощью кривых Безье удается хорошо приближать самые разные кривые. Например, можно показать, что четверть окружности можно приблизить единственной кривой Безье с точностью 0,000272*R. Для наглядности: если нужно приблизить четверть окружности радиуса 1 м кривой Безье, то максимальное отклонение при аппроксимации составит 0,272 мм. Приближение целой окружности единственной кривой Безье и даже половины окружности имеет несравненно большую погрешность, поэтому окружность приближают четырьмя симметричными кривыми Безье.

Основные графические возможности языка **PostScript** относятся к векторной графике. Процесс построения изображения, как правило, разбивается на этапы построения пути и визуализации этого пути.

Построение пути включает создание нового пути, добавления в него отрезков или кривых Безье и замыкание пути, благодаря которому путь может быть интерпретирован как линия или как область. Все пути в **PostScript**, готовые к выводу, являются замкнутыми. Различие между областью и путем проявляется в вызове операторов. Например, оператор **stroke** рисует линию вдоль построенного пути, а оператор **fill** заполняет область, ограниченную этим путем. Важным моментом при этом является определение того, какие точки являются внутренними. Для простых путей это интуитивно ясно, но для сложных случаев с самопересечением или несколькими контурами задача нетривиальна. Для определения внутренних точек в **PostScript** применяются два алгоритма. Первый, так называемое «правило ненулевого числа оборотов», состоит в том, что из испытуемой точки проводится луч в бесконечность в некотором произвольном направлении и подсчитывается сумма: если контур пересекает путь слева направо, то +1, если же справа налево, то –1. Если в итоге сумма равна 0, то точка вне контура, иначе внутри. Другое правило, «четнечет», состоит в аналогичном проведении луча и подсчете просто числа пересечений его с контуром. Если оно нечетно, точка внутри контура, если четно – снаружи.

Построение пути еще не означает появления его на устройстве вывода. Вывод включает визуализацию, когда определяется, что надо делать с этим путем: или провести вдоль него линию, или закрасить область, или использовать путь для отсечения изображения. Результат этих операций и отображается на устройстве вывода.
Для удобства выполнения многих задач в язык введена возможность применения аффинных преобразований при работе с графикой. С их помощью можно не только строить описания страниц, легко настраивающиеся на конкретные разрешение, размер страницы и т.д., но и включать в качестве элемента одной страницы преобразованное описание другой.
Аффинные преобразования поддерживаются с помощью матрицы текущих преобразований **(current transformation matrix, CTM)**, которая накапливает результаты применения на данной странице элементарных действий по масштабированию, повороту и переносу изображения. Векторная графика обладает свойством подвергаться аффинным преобразованиям без потери качества.
Несмотря на то, что огромная доля возможностей **PostScript** относится к векторной графике, в него включена поддержка и растровых изображений. Хотя явной поддержки растровых форматов файлов нет, язык включает операции описания растровых изображений с помощью обычных, несжатых битовых карт с 24-битной глубиной цвета.

На выбор операционной системы для реализации интерпретатора **PostScript** большое влияние оказал тот факт, что графическая система **GDI**, входящая в состав ядра **Windows**, построена практически на тех же понятиях и принципах, что и язык **PostScript**. В частности, рисование в **GDI** основано на использовании траекторий и областей, которые являются практически тем же самым, что и пути и области в **PostScript**. В **GDI** поддерживается рисование линий и кривых Безье. Известно также, что **GDI** поддерживает рисование и некоторых эллипсов (с осями, параллельными осям координат), что не может быть реализовано в **PostScript**. Однако, что очень важно, **GDI** имеет возможность аппроксимации траекторий кривыми Безье, а также кривых Безье – отрезками, что прочно связывает ее с графикой **PostScript**.
Дальнейшее изучение графических систем **GDI** для разных версий **Windows** показало, что возможности одних версий существенно отличаются от других. Так, возможности **GDI** в составе ядер **Windows 9x** являются подмножеством возможностей **GDI** ядра **Windows 2000**. а также всех **Windows** класса **NT**. Например, в **GDI Windows 2000** встроена поддержка так называемых «мировых преобразований», которые являются полным аналогом преобразования пользовательских координат в координаты устройства с помощью матрицы **CTM**. В **GDI Windows 9x** такая возможность не поддерживается. Другие отличия связаны, например, с рисование различных типов сочленения отрезков, что имеет большое приложение в дизайне и полиграфии. **Windows 9x** не поддерживает многие типы косметических перьев. Надо отметить, что в **PostScript** нет понятия «перьев», все свойства, влияющие на вывод графики, хранятся только в графическом состоянии.
Кроме того, реализация **GDI** в ядре **Windows 2000** имеет несколько более быстрые алгоритмы, чем в **Windows 9x**. Причины этого полностью не выяснены, но факт был установлен сравнением замеров времени работы интерпретатора над одним и тем же файлом в разных операционных системах (см. таблицу экспериментальных данных).
Таким образом, даже в современных версиях **Windows** графическая библиотека **GDI** не дотягивает по возможностям до стандарта **PostScript**. Кроме того, различные версии трудно даже сравнивать, поскольку у них слишком непохожие уровни. Поэтому при разработке интерпретатора встал вопрос, какую реализацию библиотеки **GDI** взять за основу.
Чтобы расширить сферу возможного применения данного интерпретатора, было принято решение реализовать графические возможности интерпретатора, опираясь на как можно меньший уровень **GDI**, дополнительно реализовывая необходимые алгоритмы. Так, одной из первых добавленных возможностей, было использование матрицы **CTM** для реализации аффинных преобразований.
Согласно принципам **PostScript**, интерпретация программы не должна в большинстве своем зависеть от устройства, поэтому для каждого устройства вывода можно выделить минимальный набор графических операций, который оно должно поддерживать. В него входят: штрихование пути, заливка области, определение, какая часть изображения будет выведена на устройство при использовании пути отсечения. Поэтому в интерпретаторе описан класс, представляющий самые общие свойства устройств вывода. Реализация этого класса включает виртуальные методы, представленные в минимальном наборе возможностей. Предполагается использовать этот класс как общего предка для всех классов специальных устройств вывода. Например, на основе этого класса создана реализация устройства вывода в графическое окно (дисплей) и «устройство», сохраняющее построенное векторное изображение в виде растра в файл (формат **BMP**). На основе этой технологии нетрудно создать «устройство-конвертер», которое будет преобразовывать описание страницы на **PostScript** в программу на другом языке описания страниц.

6. ЭКСПЕРИМЕНТАЛЬНЫЕ ДАННЫЕ.

После написания интерпретатора была проведена серия экспериментов с целью проверки возможностей программы по работе с различными файлами и измерения скорости. С помощью известных средств работы с векторной графикой (например, применялся **CorelDRAW** 9 и 10, а также входящий в состав пакета программ **Corel Tracer**) было создано несколько файлов различного размера, содержащих описание графических страниц на языке

Adobe PostScript. Их визуализация была осуществлена с помощью созданного интерпретатора. Результаты тестов приведены в таблице. В первом столбце указана используемая ОС, во втором – частота используемого процессора (в МГц), во третьем столбце – размер ps-файла в килобайтах. В четвертом столбце приведено общее время интерпретации файла и, наконец, в последнем – приведенная скорость интерпретации, вычисленная по формуле:

$$Скорость = \frac{Размер\ файла}{Общее\ время}$$

ОС	Частота процессора	Размер файла (кБ)	Общее время (с)	Скорость (кБ/с)
Win2000	366	59 226	72,263	819,6
Win2000	433	181	0,240	754,2
Win2000	433	5 776	6,709	860,9
Win2000	433	14 922	16,144	924,3
Win98	366	181	0,420	430,9
Win98	433	181	0,350	517,1
Win98	433	14 922	22,340	668,0

Из таблицы видно, что время обработки очень больших изображений растет не строго линейно. При больших объемах данных увеличивается скорость обработки. Частично это связано с тем, что при чтении программы используется буфер (в текущей версии его объем установлен 512 кБ), который заполняется при чтении с жесткого диска параллельно интерпретации, которая в полной мере использует процессор. На основе приведенных данных можно сделать предположение, что время обработки файла размером порядка 1 Гб (что является почти недостижимой границей для реальных файлов описания графических страниц) составляет около 20 минут.

Несмотря на то, что мы реализовали не все идеи, связанные с повышением производительности, наш интерпретатор существенно превосходит по скорости последнюю версию популярного графического пакета ACDsee. Так, ps-файл размером ..... Кбайт ACDsee отображает за ... сек.

Об авторах

Соловьев Александр Геннадьевич **–** студент 4 курса факультета ВМК Нижегородского государственного университета им. Н.И. Лобачевского.

Шишалов Михаил Викторович – студент 4 курса факультета ВМК Нижегородского государственного университета им. Н.И. Лобачевского.

тел. (8312) 50-69-35