САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Гудиев Артур Владимирович

Реализация графической части интерпретатора языка PostScript

Курсовая работа

Зав. кафедрой:
д.ф.-м.н., проф. А.Н. Терехов

Научный руководитель:
к.ф.-м.н. Д.Ю. Булычев

Санкт-Петербург

2014

Оглавление

Введение

Язык PostScript – интерпретируемый графический язык программирования, который создавался с целью представления цифровой графики в машинонезависимой форме [?]. PostScript предоставляет возможности вывода векторной и растровой графики, а также печати текста. Интерпретатор PostScript считывает программу и выводит результат на экран или принтер.

Интерпретатор PostScript оперирует сущностями, которые называются объектами PostScript. У каждого объекта есть такие свойства, как тип, значение и атрибуты. У интерпретатора PostScript есть своя среда исполнения, которая состоит из стеков, виртуальной памяти и графической среды исполнения. Виртуальная память представляет собой хранилище для значений объектов PostScript, а сами объекты находятся в стеках. Графическая среда выполнения описывает графическое состояние исполняемой программы.

Графическое состояние содержит такие параметры, как ширину линии, цвет, текущий путь, текущую матрицу преобразования и ограничивающий путь. Применяя операторы рисования и построения пути, можно нарисовать линии, дуги и кривые Безье [?], залить заданным цветом внутреннюю область пути, ограничить область рисования, преобразовать текущую систему координат.

В языке PostScript есть графические операторы, которые действуют на графическое состояние. Такие операторы могут, например, поменять масштаб рисования, задать цвет или ширину линии. Кроме того есть операторы, сохраняющие текущее графическое состояние, в котором хранятся основные графические параметры. По сохраненному состоянию можно задать текущее графическое состояние таким, каким оно было в момент сохранения.

На сегодняшний день существует большое количество платформ, у каждой из которых есть свои отличительные особенности. Реализация интерпретатора

PostScript для каждой платфромы довольно затруднительна, поэтому необходимо обеспечить переносимость интерпретатора.

Java – объектно-ориентированный язык программирования. Программы на Java переводятся компилятором Java (javac) в специальный байт-код Java. Виртуальная машина Java (сокращенно Java VM, JVM) – основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java интерпретирует байт-код. В настоящий момент виртуальная машина Java широко распространена. Следовательно, программа, написанная на языке Java, может быть исполнена на большом количестве платформ вне зависимости от архитектуры компьютера. Язык Java предоставляет основные графические возможности для рисования.

Целью курсовой работы является реализация графической части интерпретатора языка PostScript. Необходимо реализовать на языке Java графическую среду исполнения для интерпретатора программ PostScript. Интерпретатор должен поддерживать основные графические операторы, отображать графическое состояние на экране согласно спецификации.

1 Общие сведения о языке PostScript

Язык PostScript – это динамический интерпретируемый язык программирования, который содержит стандартное множество типов данных (числа, массивы и строки), основные элементы управления (условия, метки и процедуры), а также некоторые необычные особенности, например, словари.

Язык PostScript имеет следующие графические возможности, содержащиеся в среде языка:

- изображения произвольных графических объектов, построенных из отрезков, дуг и кубических кривых;
- графические операторы, позволяющие обрисовывать формы линиями произвольной ширины, залить область любым цветом или использовать ее в качестве ограничивающего пути;
- координатную систему, которая поддерживает все комбинации линейных преобразований.

Интерпретатор языка PostScript действует, исполняя последовательность объектов. Объекты, которые нужно исполнить, могут придти из двух источников:

- Поток символов может анализироваться согласно синтаксическим правилам языка, при этом символы группируются в токены, а затем из одного или более токенов создаются объекты. Как только объект проанализирован, он немедленно исполняется.
- Объекты, ранее сохраненные в массиве, могут быть исполнены последовательно. Такой массив называется *процедурой*.

1.1 Модель данных

Все данные, доступные программе PostScript, существуют в форме объектов. У каждого объекта есть *тип* и *значение*.

Объекты бывают *простые* и *сложные*. Простой объект является константой, его тип, значение и атрибуты неизменно соединены вместе и не могут быть изменены. У сложных объектов значения хранятся отдельно от самих объектов.

Важное отличие между *простыми* и *сложными* объектами проявляется при операциях копирования объектов. При копировании *простого* объекта все его части копируются вместе. При копировании *сложного* объекта *значение* не копируется, вместо этого исходный и скопированный объекты разделяют одно значение. Следовательно, если значение некоторого исходного объекта изменилось, то оно изменилось у всех объектов, разделяющих одно значение вместе с исходным.

Типы данных и объекты

Объекты в PostScript бывают простые и сложные. Ниже приведены виды простых и сложных объектов в PostScript:

Простые объекты	Сложные объекты
boolean	array
integer	dictionary
mark	gstate
name	save
null	string
operator	
real	

Числа в языке PostScript бывают:

- знаковые целые (*integer*), такие как `123, -76, 0, +17`;
- вещественные (*real*), такие как `-.002, 56.7, 123.6e10`.

Для логических выражений поддерживаются *логические* объекты со значениями `true` и `false`.

Объекты *операторы* (*operator*) представляют некоторые встроенные в язык действия. *Операторы* связаны со своими именами (например, `fill, gsave и rlineto`) в стандартном словаре `systemdict`.

Любой токен, состоящий из стандартных символов, который нельзя рассматривать как число (например, `"abc"`, `"Offset"`, `"23A"`) является *именем* или *именным объектом* (*name*). Символ /, являющийся префиксом некоторого имени, представляет *буквенное имя* (*literal name*), например `"/variable"`.

Массивом (*array*) является последовательность объектов PostScript, заключенная между "["и "]", например `" [ 123 /abc (xyz) ]"`.

Похожий синтаксис есть у *процедур* (*procedure*), которая является *исполняемым массивом. Процедура* — это последовательность объектов, заключенная между "{"и "}" , например, `"{ add 2 div }"`.

Последовательность пар, состоящих из ключа и значения, которая заключена между "«"и "»" , определяет словарь в PostScript, например, `"« (number) 17 (string) (world) (array) [1 2] »"`

Для сохранения состояния памяти интерпретатора существуют объекты *сохранения* (*save*), которые создаются оператором `save` и с помощью которых можно восстановить состояние памяти оператором `restore`. Также есть объект *графического состояния* (*gstate*), предназначенный для использования графических возможностей языка PostScript.

1.2 Структура памяти интерпретатора

Данные в языке PostScript представлены в виде объектов. Важное место в языке занимают стеки и виртуальная память, поскольку все объекты хранятся на стеках, а виртуальная память предоставляет место для хранения значений сложных объектов.

1.2.1 Стеки

Интерпретатор PostScript управляет стеками, которые представляют исполняемое состояние программы.

- Стек операндов содержит произвольные объекты PostScript, являющиеся операндами или результатами некоторых исполняемых операторов.
- Стек словарей содержит только словари. В нем содержатся три стандартных словаря `systemdict`, `userdict` и `globaldict`.
- Стек исполнения содержит исполняемые объекты, находящиеся в промежуточной стадии исполнения.
- Графический стек хранит только объекты графического состояния.

1.2.2 Управление памятью

В языке PostScript предусмотрено хранилище для значений *сложных* объектов — *виртуальная память*. *Сложный* объект имеет ссылку на свое *значение*. *Виртуальная память* подразделяется на *локальную* и *глобальную*. Основным отличием является то, что операторы сохранения `save` и восстановления `restore` состояния памяти влияют только на *локальную виртуальную память*. Для освобождения памяти, выделенной для объектов, которые уже не используются, используется сборщик мусора.

1.3 Графика

Воздействовать на графическую состояние программы можно с помощью графических операторов. В языке PostScript графические операторы можно разделить на три группы: операторы построения пути, рисующие операторы и операторы координатной системы и матрицы преобразования.

Интерпретатор PostScript неявно поддерживает *текущую страницу*, которая накапливает результаты работы рисующих операторов. В начале исполнения программы *текущая страница* пуста. Как только некоторый рисующий оператор исполняется, он кладет соответствующие метки на *текущую страницу*. При перекрытии каждое новое изображение загораживает все старые. Как только *текущая страница* сформирована, вызов оператора *showpage* изображает все накопленные результаты графических операторов на экране и очищает *текущую страницу*.

Общий алгоритм рисования для программ PostScript можно описать следующим образом:

1. формирование пути с помощью операторов построения пути;
2. задание некоторых неявных параметров при необходимости;
3. вызов рисующих операторов.

Интерпретатор PostScript поддерживает внутреннюю структуру данных — *графическое состояние*, хранящую графические параметры управления, такие как текущую позицию `position`, текущую матрицу преобразования `cTM`, текущий путь `path`, ограничивающий путь `clipping path`, цвет `color`, ширину линии `line width`, форму конца линии `line cap`, форму соединения отрезков `line join` и параметр пунктирности линии `dash pattern`.

Язык PostScript определяет декартову систему координат, которую программы могут использовать, чтобы определить расположение произвольной точки на странице. *Текущая позиция* представляет собой пару вещественных координат. Многие графические операторы (например, `lineto, moveto, rcurveto`) неявно используют *текущую позицию* для рисования.

Систему координат можно изменять с помощью линейных преобразований. Линейные преобразования системы координат PostScript представлены в виде *текущей матрицы преобразования* `cTM`, представленной в виде массива из шести чисел $[\ a, b, c, d, t_x, t_y\ ]$:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Такая матрица определяет относительные координаты, которые выражаются через абсолютные:

$$x' = ax + cy + t_x$$

$$y' = bx + dy + t_y$$

В языке PostScript определены такие линейные преобразования, как перенос, поворот и масштаб. Каждое преобразование определяется соответствующей матрицей.

Матрица сдвига на вектор (t_x, t_y):

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Матрица изменения масштаба, в которой параметры s_x и s_y определяют мас-

штаб по осям x и y соответственно:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Матрица поворота на угол θ:

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Операторы переноса `translate`, поворота `rotate` и масштаба `scale` изменяют *текущую матрицу преобразования*, так что новая матрица является произведением матрицы соответствующего преобразования и исходной матрицы:

$$\texttt{новая матрица} = \texttt{преобразование} \times \texttt{исходная матрица}$$

Оператор `transform`, принимающий один параметр — массив из шести чисел,— переопределяет новую *матрицу преобразования*.

В языке PostScript есть объекты *пути*, характеризующие графические объекты. Программы используют *пути* для рисования линий, определения форм закрашиваемой области и границ для ограничения области рисования. Объект *пути* состоит из связанных подпутей, построенных из прямых линий, кубических кривых и дуг эллипсов. Изначально есть только один подпуть, к которому могут добавляться новые элементы, не нарушающие его связанность. При попытке добавления элемента, нарушающего связанность, создается новый подпуть, и все новые элементы добавляются уже к нему, таким образом все подпути объекта *пути* остаются связанными. В графическом состоянии языка PostScript есть два объекта пути: *текущий путь* и *ограничивающий путь*.

Текущий путь можно изменить, используя операторы перемещения текущей позиции (`moveto` и `rmoveto`), присоединения к пути отрезка (`lineto` и `rlineto`), различных дуг (`arc` и `rarc`), кубических кривых Безье (`curveto` и `rcurveto`), а также оператор замыкания текущего связанного подпути `closepath`.

Ограничивающий путь используется с помощью оператора пересечения текущего пути и текущего ограничивающего пути `clip` и оператора замены текущего пути копией текущего ограничивающего пути `clippath`.

Ширина линии представляет собой число и может быть изменена с помощью оператора `setlinewidth`.

Форма конца линии также представлена числом, принимающим три значения: 0, 1 и 2. Присвоить значение можно с помощью оператора `setlinecap`. Каждому числу соответствует своя форма конца:

0 — стыковой конец;

1 — круглый конец;

2 — выдвинутый прямоугольный конец.

Форма соединения отрезков в PostScript характеризуется числом, принимающим три значения: 0, 1 и 2. Присвоить значение можно с помощью оператора `setlinejoin`. Для каждого ччисла определена своя форма соединения:

0 — соединение под углом 45°;

1 — круглое соединение;

2 — скошенное соединение.

Рис. 1: Формы параметров конца линии

Параметр пунктирности задается с помощью двух элментов: массива, в котором числа поочередно показывают, сколько пикселей нужно заштриховать, а сколько пропустить, и начального сдвига, задаваемого целым числом.

Непосредственно рисованием занимаются операторы прорисовки `stroke` и операторы заливки цветом `fill` по правилу ненулевого количества поворотов (Nonzero winding number rule) и `eofill` по правилу четный-нечетный (Even-odd rule) [**?**]. Операторы рисования неявно используют такие параметры, как ширину линии и цвет. Самым важным неявным параметром является *текущий путь*, который изменяется при исполнении рисующего оператора.

Для сохранения и восстановления графического состояния предусмотрены

Рис. 2: Формы параметров соединения отрезков

операторы `gsave` и `grestore`, которые соответственно кладут на графический стек копию графического состояния и восстанавливают сохраненное значение по копии.

1.4 Пример программы на языке PostScript

Для примера рассмотрим простую программу, рисующую прямоугольники, на языке PostScript:

Листинг 1: Простая программа на языке PostScript

```
% transform inches to pixels
/inch {72 mul} def
```

Рис. 3: Примеры параметров пунктирности линии

```
% construct a rectangle
/box
{newpath moveto
1 inch 0 rlineto
0 1 inch rlineto
−1 inch 0 rlineto
closepath}
def

% fill a shape with color
/fillbox {setgray fill} def

% draw three rectangels
3.5 inch 4.5 inch box
0 fillbox
3.75 inch 5 inch box
.4 fillbox
4 inch 5.5 inch box
.8 fillbox
showpage
```

Данная программа закрашивает три прямоугольника разными цветами:

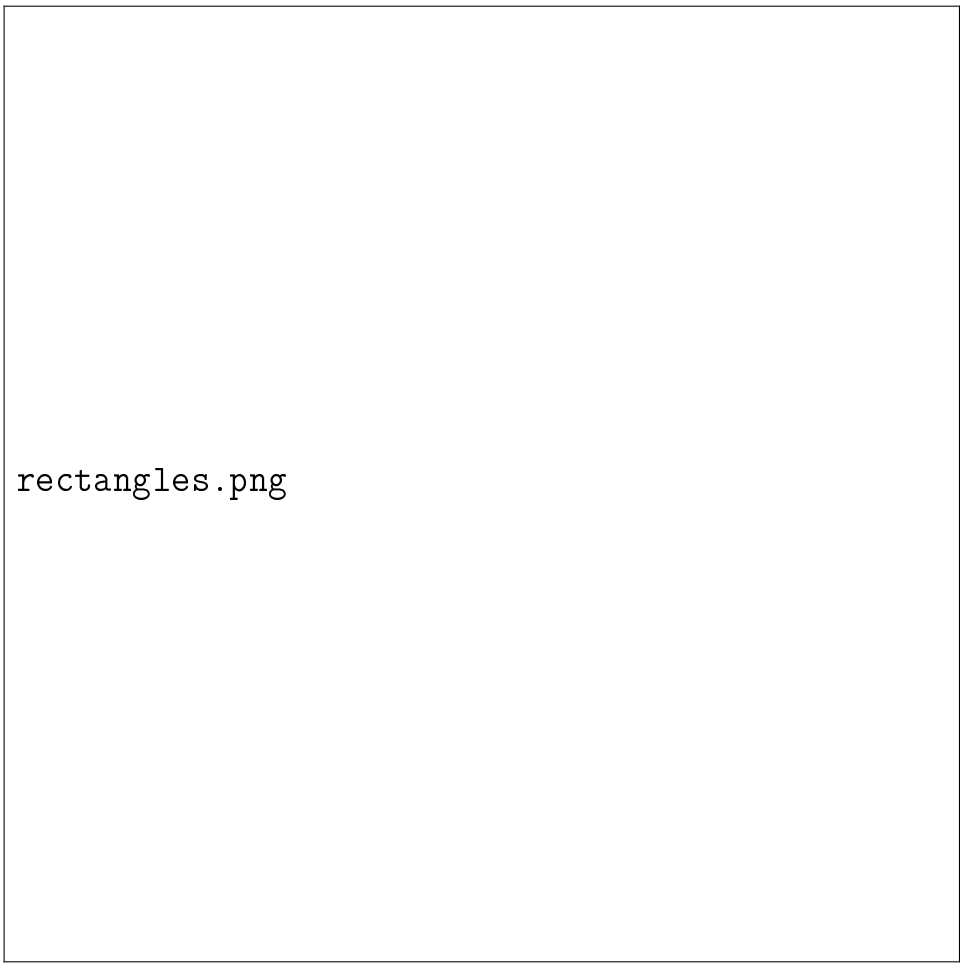

Рис. 4: Результат работы программы

2 Реализация графической среды исполнения

При реализации графической среды исполнения были использованы стандартные библиотеки языка Java: `jawa.awt.geom`, `jawa.awt.image` и `java.swing`. С помощью класса GeneralPath из библиотеки `jawa.awt.geom` были реализованы класс пути `PSPath` и операторы построения пути. С помощью класса JFrame из библиотеки textttjava.swing и класса BufferedImage из `jawa.awt.image` были реализованы рисующие операторы.

`GState` характеризует графическую среду исполнения интерпретатора PostScript. В классе хранятся такие параметры, как текущий путь `currentPath`, ограничивающий путь `clippingPath`, текущая матрица преобразования системы координат `cTM` и графические настройки `graphicsSettings`.

Текущий путь и ограничивающий путь характеризуются классом `PSPath`, реализованным с помощью стандартного класса `generalPath` из библиотеки `java.awt.geom`. По объекту PSPath можно получить ограничивающий прямоугольник `getBBox()`, сделать замкнутым текущий подпуть `closePath()` и добавить в путь отрезок `addLine()`, кривую Безье `addCurve()` и дугу эллипса `addArc()`.

Текущая матрица преобразования описывается классом `TransformMatrix`. Сама матрица представлена массивом из шести чисел в виде объекта класса `PSObject`. По экземпляру `TransformMatrix` можно изменить масштаб `scale()`, сделать параллельный перенос `translate()` и повернуть `rotate()` систему координат, а также взять обратную матрицу `getInverseMatrix()` и по заданным относительным координатам получить абсолютные координаты `transform()`.

Графические настройки представлены классом `GraphicsSettings`, содержащим такие параметры, как цвет, ширина линии, параметры соединения отрезков и параметры пунктирной линии.

2.1 Отображение на экране

С наглядным представлением графической среды исполнения на экране связаны такие классы, как `PSImage`, `PSFrame` и `PSDrawer`. Класс `PSImage`, реализованный с помощью стандартного класса `java.awt.image.BufferedImage`, представляет отображение графического состояния на экране. Класс `PSDrawer` непосредственно отрисовывает графическое состояние. Объект `PSDrawer` хранит экземпляр класса `PSFrame`, наследуемого от стандартного класса `javax.swing.JFrame` и отвечающего за окно, в котором изображается графическое состояние.

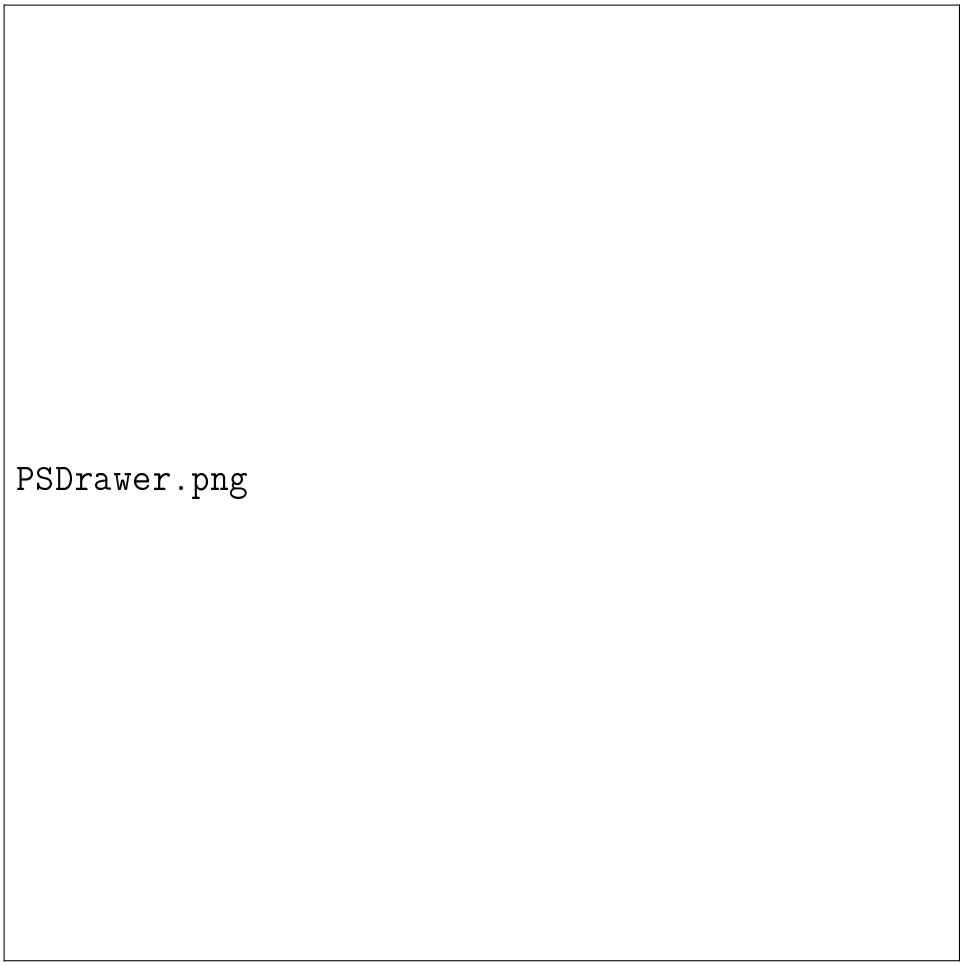

Рис. 5: Диаграмма классов отображения

2.2 Графические операторы

Графические операторы, являющиеся наследниками абстрактного класса `AbstractGraphicOperator`, непосредственно воздействуют на графическое состояние. Операторы, похожие по поведению, можно объединить в группы. Графические операторы разделяются на группы операторов графического состояния, построения пути и рисования.

Среди операторов графического состояния есть операторы сохранения `GSaveOp()` и восстановления `GRestoreOp()` графического состояния, операторы задания цвета `SetRgbColorOp()`, ширины линии `SetLineWidth()` и параметров пунктирной линии `SetDashOp()`.

В операторы построения пути входят такие операторы, как операторы построения отрезка `LineToOp()` и `RLineToOp()`, дуги эллипса `ArcOp()` и `ArcnOp()`, кривой Безье `CurveToOp()` и `RCurveToOp()`, а также операторы перемещения текущей точки `MoveToOp()` и `RMoveToOp()`. Кроме того, можно взять ограничивающий прямоугольник пути `PathBBoxOp()`, задать новый текущий путь `NewPathOp()` и начальный ограничивающий путь `InitClipOp()`, изменить ограничивающий путь `ClipPathOp()`, а также замкнуть текущий подпуть в текущем пути `ClosePathOp()`.

Среди операторов рисования есть операторы рисования контура пути `StrokeOp()`, заливки области пути `FillOp()` и `EoFillOp()`, а также заливки области заданного прямоугольника `RectFillOp()`.

Заключение

В рамках курсовой работы разработана графическая среда исполнения для интерпретатора языка PostScript. В графической среде выполнения хранятся такие параметры, как текущая точка, текущий путь, ограничивающий путь, цвет, матрица преобразования системы координат и ширина линии. При этом выполнены следующие требования:

- Реализованы основные графические операторы.
- Поддерживаются операторы сохранения графического состояния.
- Есть возможность визуального представления графического состояния на экране.
- Графическая среда исполнения интерпретатора PostScript реализована на языке Java.

Список литературы

[1] PoscScript Language Reference. Third edition, 1999
`http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`

[2] Д. Роджерс, Дж. Адамс. Математические основы машинной графики. 2001.