УДК 004.514

СПОСОБЫ КОНВЕРТАЦИИ ТЕКСТА В PDF-ДОКУМЕНТАХ В ФОРМАТ SVG (МАСШТАБИРУЕМОЙ ВЕКТОРНОЙ ГРАФИКИ)

Е. И. ЮРЧИШКО
(Представлено: канд. техн. наук, доц. В. М. ЧЕРТКОВ)

Описаны особенности форматов документов PDF и SVG с акцентом на представление текстового содержимого и векторной графики. Приводится функциональная структура возможного механизма конвертации текста из PDF-документов в формат двумерной векторной графики SVG. Описывается принятая концепция обработки метаданных текста, анализа шрифтов и построения векторных контуров глифов с сохранением позиционирования и визуальных атрибутов. Представлена диаграмма потоков данных для возможного программного решения поставленной задачи конвертации.

Введение. форматы документов играют ключевую роль в обмене информацией, хранении данных и представлении контента. Среди множества существующих форматов особое место занимают PDF (Portable Document Format), разработанный компанией Adobe, стал де-факто стандартом для распространения документов, поскольку обеспечивает сохранение внешнего вида независимо от платформы и устройства [1].

SVG, в свою очередь, представляет собой XML-основанный векторный формат, предназначенный для отображения двумерной графики в вебе и других средах, где важны масштабируемость и редактируемость [2]. В определённых сценариях возникает необходимость преобразования текстового содержимого из PDF в SVG – например, при визуализации документов в веб-приложениях, при подготовке графики для интерактивных интерфейсов или при создании редактируемых версий печатных материалов.

Постановка задачи. В PDF-документе текст может храниться по-разному: как обычный текстовый поток с привязкой к шрифтам, как векторные контуры (например, после экспорта из графических редакторов), или даже как растровое изображение. Для успешного извлечения текста и его последующей конвертации в SVG необходимо определить, в каком виде текст присутствует в исходном документе. Если текст сохранён как текстовый объект с метаданными о шрифтах и позиционировании, задача значительно упрощается. Однако если текст представлен в виде векторных контуров без семантической информации, процесс становится сложнее и может потребовать применения методов компьютерного зрения или оптического распознавания символов (OCR).

SVG, в отличие от PDF, является чисто векторным форматом, ориентированным на описание графических примитивов: линий, кривых, прямоугольников, текстовых блоков и т.д. Текст в SVG может быть представлен либо как элемент *<text>* с указанием шрифта, размера и координат, либо как набор векторных путей *(<path>),* описывающих форму каждого символа. Преобразование текста из PDF в SVG может преследовать разные цели: если требуется сохранить возможность редактирования текста, предпочтительно использовать *<text>*; если же важна точная визуальная реплика без зависимости от системных шрифтов, то текст следует конвертировать в векторные пути.

Для реализации механизма конвертации текста из PDF в SVG целесообразно использовать языки программирования высокого уровня, такие как Python, JavaScript, Java или C#. Эти языки обладают богатыми экосистемами библиотек и фреймворков, которые значительно упрощают работу с документами

и графикой. Python, например, предлагает такие инструменты, как *pdfplumber*, *PyMuPDF* (также известный как *fitz*), *pdfminer.six* для извлечения текста и метаданных из PDF, а также *svgwrite*, *cairosvg*, *reportlab* для генерации SVG. JavaScript, особенно в контексте Node.js, предоставляет библиотеки вроде *pdf-lib*, *pdf2json*, *svg.js*, *canvg*, что делает его удобным выбором для веб-ориентированных решений. Java и C# также имеют подходящие библиотеки – *Apache PDFBox* и *iText* для работы с PDF, а также *Batik* и *SkiaSharp* для генерации SVG.

В случае, если PDF содержит текст, уже преобразованный в векторные контуры (например, при экспорте из Adobe Illustrator), задача сводится к извлечению этих контуров и их переносу в SVG. Библиотека PyMuPDF позволяет получать векторные команды (линии, кривые Безье) из PDF-страницы, что упрощает прямое преобразование в SVG-пути. Здесь важно сохранить иерархию объектов и стили (цвет заливки, обводки, прозрачность), чтобы визуальное представление осталось неизменным.

Особую сложность представляет обработка многостраничных PDF-документов. SVG, в отличие от PDF, не имеет встроенной поддержки нескольких страниц. Поэтому при конвертации многостраничного PDF в SVG возможны два подхода: либо генерировать отдельный SVG-файл для каждой страницы, либо объединить все страницы в один SVG-документ с использованием групп *(<g>)* и смещений по вертикали или горизонтали. Первый подход проще и чаще используется, особенно в веб-приложениях, где каждая страница может загружаться по отдельности.

Ещё одним важным аспектом является обработка шрифтов. В PDF шрифты могут быть встроены полностью или частично (subset), что означает, что в документе присутствуют только те глифы, которые используются в тексте. При конвертации в SVG необходимо либо включить эти шрифты в виде векторных путей, либо обеспечить их доступность в среде отображения. Веб-стандарты позволяют встраивать шрифты в SVG с помощью элемента <*defs*> и @*font-face*, однако это увеличивает размер файла и не всегда поддерживается всеми рендерерами. Поэтому наиболее надёжным решением остаётся преобразование текста в пути.

При реализации механизма конвертации на языке высокого уровня следует также учитывать производительность и масштабируемость. Обработка больших PDF-документов может быть ресурсоёмкой, особенно если требуется точное воспроизведение векторной графики. Оптимизация может включать в себя параллельную обработку страниц, кэширование шрифтов, минимизацию SVG-выхода и использование потоковой обработки вместо загрузки всего документа в память.

Механизм конвертации в формат SVG на языке программирования C#. Выбор данного языка программирования актуален в экосистеме .NET, где разработчики часто сталкиваются с задачами обработки документов в корпоративных приложениях, системах электронного документооборота, а также в решениях, ориентированных на Windows-платформу.

В экосистеме .NET наиболее популярными библиотеками для работы с PDF являются *iTextSharp* (сейчас переименована в *iText 7 for .NET*), *PdfiumViewer*, *PDFsharp*, а также коммерческие решения вроде *Aspose.PDF for .NET* и *Syncfusion PDF Library*. Для задачи извлечения не только текста, но и точных координат его размещения, а также информации о шрифтах, наиболее подходящей является *iText 7* или *Aspose.PDF*, так как они предоставляют низкоуровневый доступ к содержимому страниц.

Например, с использованием *iText 7* можно реализовать персонализированный обработчик событий IEventListener, который перехватывает все текстовые операции при рендеринге страницы [3]. Внутри обработчика события Event можно получить объекты типа TextRenderInfo, содержащие:

- Строку текста (GetText()).
- Координаты базовой линии (GetBaseline()).
- Информацию о шрифте (GetFont()).
- Цвет текста (GetFillColor()).

Эти данные позволяют точно воссоздать положение и внешний вид текста в целевом SVG-документе.

В качестве альтернативы, *Aspose.PDF* предлагает более высокоуровневый API: через вызов объекта TextFragmentAbsorber можно извлечь все фрагменты текста на странице вместе с их параметрами, такими как: позиция, размер шрифта, наименование шрифта, колер начертания и другими свойствами.

Одна из ключевых проблем при конвертации – это корректная обработка шрифтов. В PDF шрифты могут быть:

- Стандартными (например, Helvetica, Times-Roman).
- Встроенными (embedded), в том числе частично (subset).
- Не встроенными (тогда рендерер полагается на системные шрифты).

Если шрифт не встроен в PDF, то при конвертации в SVG возникает риск того, что на целевом устройстве он будет отображён иначе или заменён на шрифт по умолчанию, что нарушит визуальную целостность документа. Чтобы избежать этого, рекомендуется конвертировать текст в векторные контуры (glyph paths).

В C# для получения контуров глифов можно использовать *SharpFont* – обёртка над библиотекой *FreeType*, позволяющая загружать TTF/OTF-файлы и извлекать векторные пути символов; *System.Windows.Media* (в WPF) – через GlyphTypeface и GetGlyphOutline(), но это требует наличия WPF и работает только в операционной системе Window;. *SkiaSharp* – кроссплатформенная библиотека от Google, которая предоставляет доступ к шрифтам и позволяет рендерить текст в векторные пути через SKPath.

Практически, наиболее универсальным решением является SkiaSharp, особенно если приложение должно работать на Linux или macOS. С его помощью можно.

Для представления текста в SVG существуют два основных подхода: текстовые элементы *(<text>)* – компактны, редактируемы, но зависят от доступности шрифтов, векторные пути *(<path>)* – не зависят от шрифтов, точно воспроизводят форму, но увеличивают размер файла и делают текст нечитаемым для поиска или копирования.

Если цель – максимальная точность, предпочтительно вручную формировать SVG с использованием XmlWriter, вставляя <*path*> для каждого символа. Координаты путей берутся из SKPath, а трансформации (масштаб, поворот) применяются на основе данных из PDF (TextMatrix).

Важно учитывать систему координат: в PDF начало координат находится в нижнем левом углу, тогда как в SVG – в верхнем левом. Поэтому при переносе координат необходимо инвертировать ось Y по следующей формуле:

$$svgY = pageHeight - pdfY.$$

Ниже приведен пример диаграммы потоков данных на языке UML, иллюстрирующий систему (приложение, реализующее функционал конвертера текста из PDF-документа в формат SVG), как совокупность процессов, потоков данных и их взаимодействие с внешними сущностями. Внешняя сущность представлена в виде пользователя системы, который в качестве потока данных предоставляет файл формата PDF, который используется в процессе парсинга (происходит извлечение текстовых объектов, координат, шрифтов, цветов). Далее структурированные данные о тексте попадают в хранилище. Ранее извлеченная информация о шрифтах (или одном шрифте) используется в процессе анализа и обработки. Данный процесс определяет, встроен ли шрифт в PD. Если шрифт недоступен – инициирует конвертацию в векторные контуры, при необходимости извлекает встроенные шрифты как временные TTF/OTF. Это два наиболее распространённых формата цифровых шрифтов, используемых в операционных системах, графических редакторах, веб-браузерах и программном обеспечении для отображения текста. Оба формата описывают визуальное представление символов (букв, цифр, знаков препинания и т.д.) с помощью векторной графики. Далее происходит непосредственно генерация векторных кривых и массив из полученных данных используется для формирования итогового документа SVG-формата, который получает пользователь системы.

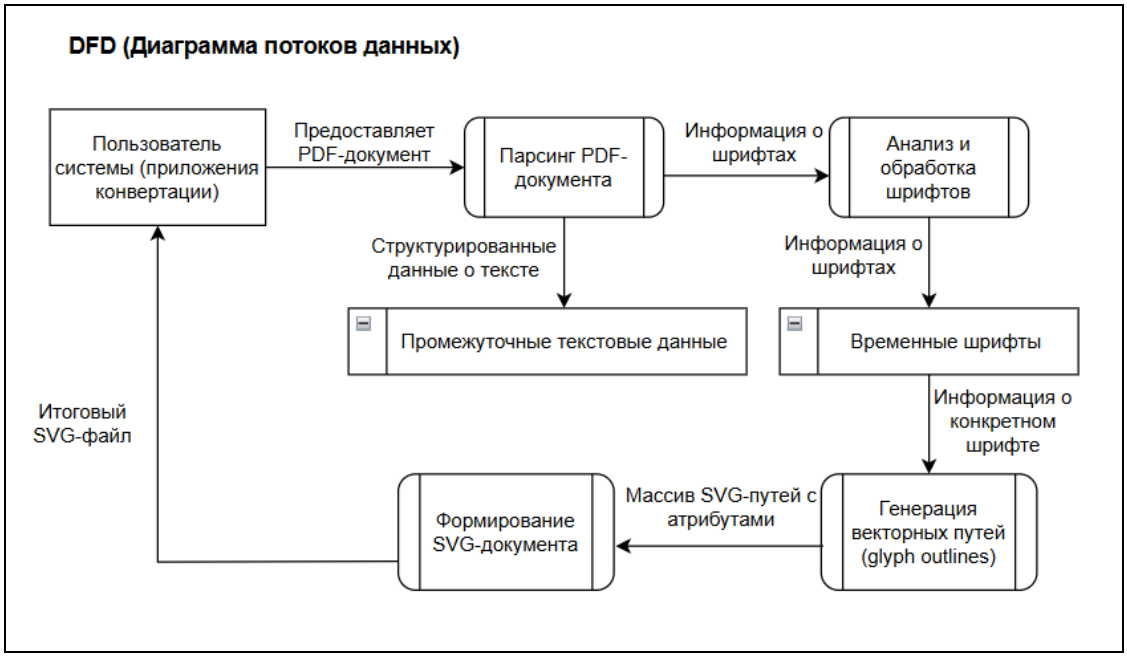

Рисунок 1. – Диаграмма потоков данных для приложения-конвертера

Для практического применения механизма преобразования текста из PDF в SVG особенно эффективным решением является создание настольного приложения с графическим интерфейсом, ориентированного на пользователей операционной системы Windows. В экосистеме .NET такой задаче идеально соответствует технология Windows Presentation Foundation (WPF) – современный фреймворк для построения богатых клиентских приложений с высокой степенью кастомизации интерфейса и глубокой интеграцией с другими компонентами платформы [4].

WPF предоставляет не только гибкие средства визуального проектирования, но и мощную архитектурную основу, позволяющую отделить логику обработки данных от представления. Это особенно важно в контексте конвертации документов, где необходимо управлять сложными процессами – от выбора входного файла до поэтапной обработки страниц, анализа шрифтов и генерации векторного вывода. Благодаря поддержке асинхронного программирования, привязок данных (data binding) и команд, WPF позволяет реализовать отзывчивый интерфейс, который не «зависает» даже при обработке объёмных PDF-документов.

Архитектура типичного конвертера на WPF строится вокруг нескольких ключевых компонентов. Во-первых, это модуль выбора файла, реализованный через стандартный диалог операционной системы, что обеспечивает привычный пользовательский опыт. Во-вторых, ядро приложения содержит логику конвертации, инкапсулированную в отдельный сервис или статический класс, отвечающий за оркестрацию взаимодействия между библиотеками обработки PDF и генерации SVG. Такой подход способствует тестируемости и повторному использованию кода. В-третьих, интерфейс включает визуальные индикаторы прогресса и состояния – текстовые сообщения, кнопки с динамической активацией, а в продвинутых версиях – прогресс-бары или превью результата.

Заключение. Преобразование текстового содержимого PDF-документов в формат двумерной векторной графики SVG представляет собой нетривиальную, но практически значимую задачу, находящую применение в самых разных областях – от веб-визуализации и цифровых архивов и до систем электронного документооборота. В ходе исследования было упомянуто, что наиболее важно при конвертации сохранить визуальную идентичность исходного документа. Это возможно лишь при точном извлечении не только самого текста, но и его метаданных – позиции, размера, цвета, трансформаций и, что особенно важно, информации о шрифтах. В случае отсутствия гарантий доступности оригинальных шрифтов в целевой среде наиболее надёжным подходом становится конвертация текста в векторные контуры, что обеспечивает полную независимость от системных ресурсов и абсолютную точность отображения.

Языки программирования высокого уровня, и в особенности C# в экосистеме .NET, предоставляют разработчикам мощный инструментарий для решения подобных задач. Благодаря большому выбору возможных библиотек, таких как *iText 7*, *Aspose.PDF*, *PdfPig* для анализа PDF и *SkiaSharp*, *SharpFont* или встроенных средств WPF для работы с векторной графикой, становится возможным построение системы (приложения) с требующимся функционалом. При этом архитектура приложения может быть адаптирована под конкретные требования: от простого консольного утилитарного инструмента до полноценного настольного приложения с графическим интерфейсом на базе WPF.

Важно подчеркнуть, что успешная реализация механизма конвертации – это не только техническая задача, но и баланс между точностью, производительностью и удобством использования. Выбор между представлением текста как редактируемого элемента или как неизменяемого векторного пути, обработка многостраничных документов, управление памятью при работе с большими файлами, корректная обработка встроенных шрифтов – всё это требует взвешенных архитектурных решений и внимания к деталям.

ЛИТЕРАТУРА

1. Adobe Systems Incorporated. PDF Reference, Sixth Edition, Version 1.7. [Электронный ресурс] – Режим доступа: https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.7old.pdf. – Дата доступа: 15.09.2025.
2. World Wide Web Consortium (W3C). Scalable Vector Graphics (SVG) [Электронный ресурс] – Режим доступа: https://www.w3.org/TR/SVG2/. – Дата доступа: 15.09.2025;
3. iText Software. iText 7: Core and Add-ons Documentation [Электронный ресурс] – Режим доступа: https://itextpdf.com/itext-suite-net-c. Дата доступа: 16.09.2025 – 21.09.2025;
4. Microsoft. Windows Presentation Foundation (WPF) Documentation [Электронный ресурс] – Режим доступа: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/. Дата доступа: 22.09.2025.