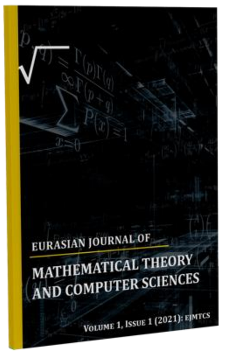

CONVERT IMAGES TO PDF IN C#

Bozorov Abdumannon
Lecturer, department of Engineering and technical support of security, University of Public Security, Republic of Uzbekistan
Shoyqulov Shodmonkul Qudratovich
Acting Associate Professor, department of Applied Mathematics, Karshi State university, Republic of Uzbekistan
https://doi.org/10.5281/zenodo.13991413

ARTICLE INFO
Received: 20th October 2024
Accepted: 24th October 2024
Online: 25th October 2024
KEYWORDS
C#, iTextSharp, image conversion, PDF, image scaling, document management automation, programming.

ABSTRACT
This article describes the development of a C# program for converting images to PDF using the iTextSharp library. Key aspects of the image conversion algorithm, as well as methods for scaling and correctly integrating them into a PDF document, are considered. The program's performance is analyzed when working with various types of images. Particular attention is paid to the advantages of using C# to solve the conversion problem, and a comparative analysis with alternative approaches in other programming languages is conducted. The article provides examples of practical application of the program in such areas as document management automation and archive creation.

INTRODUCTION

Modern digital document management tasks require the use of effective tools for converting various data formats into universal ones, such as PDF. PDF (Portable Document Format) has become a generally accepted standard for storing and exchanging electronic documents due to its platform independence, data compression capabilities, and support for information security. One of the current tasks is converting images into PDF documents for easy archiving, publishing, or exchanging files.

The C# programming language is popular for developing applications for the Windows platform, and the iTextSharp library offers extensive capabilities for working with PDF documents. Using C# in combination with iTextSharp allows you to automate the process of converting images to PDF, providing high performance and flexibility in setting parameters, which makes this approach convenient and effective[2,6].

This article discusses approaches and algorithms for converting images to PDF, analyzes the advantages of using C# and the iTextSharp library to solve this problem. It also covers issues of image scaling and program optimization for processing large amounts of data, which is especially important for archiving and document management tasks.

RESULTS and DISCUSSIONS

There are many different approaches and technologies for creating programs that convert images to PDF. One of the key factors when choosing a suitable tool is the ease of integration with other systems, the speed of operations, and the ability to process images of various formats. This section discusses the main technologies used to solve the problem of converting images to PDF, as well as their advantages and disadvantages.

C# is one of the leading programming languages for creating applications for the Windows platform. Its key advantages include high performance, powerful graphics capabilities, ease of working with objects, and access to extensive .NET Framework libraries. C# also supports the development of applications with a graphical interface, which makes it a good choice for creating utilities that convert images to PDF.

One of the most popular tools for working with PDF documents in C# is the iTextSharp library. It is a port of the iText library to Java and provides developers with extensive capabilities for creating, editing, and managing PDF files. With iTextSharp, you can not only generate PDF documents, but also add text, images, and manage the structure and format of the document[1,5].

One of the key features of iTextSharp is its flexibility and ability to process large PDF documents. This library offers tools that simplify the process of scaling images, embedding them in PDF, and optimizing the document. With support for a large number of image formats, iTextSharp becomes a convenient tool for solving image-to-PDF conversion problems in various areas, such as document management, archiving, and data publishing.

There are many different methods and tools for developing programs that convert images to PDF. Some of the main criteria for choosing a suitable solution are ease of integration with other systems, speed of task execution, and the ability to work with images of various formats. This section presents the key technologies used to perform this task, as well as their advantages and disadvantages.

C# stands out for its high performance, powerful capabilities for working with graphics, ease of use of objects, and access to the rich .NET Framework library. This language is especially suitable for creating applications with a graphical interface, which makes it an optimal choice for developing utilities that convert images to PDF. One of the most common tools for working with PDF documents in C# is the iTextSharp library. It is an adapted version of the iText library for Java and offers extensive capabilities for creating, editing and managing PDF files. Using iTextSharp, you can not only generate PDF documents, but also add text and graphic elements to them, as well as customize the structure and formatting of files. An important advantage of iTextSharp is its flexibility and ability to work with large PDF documents, which makes it a convenient tool for solving image-to-PDF conversion problems in a variety of areas[3,7].

There are several alternative technologies that can be used to perform the task of converting images to PDF:

1. PDFsharp is a C# library that provides tools for creating PDF documents with images. It has a simpler interface than iTextSharp, but offers limited functionality when working with large and complex PDF documents.
2. ImageMagick is a powerful image processing tool that supports conversion to PDF. Although it does not require programming, its integration into the application may require additional settings and configurations.

3. Adobe PDF SDK is one of the most advanced tools for working with PDF, providing a wide range of features. However, its high cost and complexity of use may limit its use in small projects.

In addition to C#, image conversion to PDF can be implemented in other programming languages, such as Python, Java, and C++. For example:

- Python provides the ReportLab library, which simplifies the creation of PDF documents with images. This language is known for its ease of use, but may be slower when processing large amounts of data.
- Java uses the iText library, similar to iTextSharp in C#. Java also provides high performance, but development in this language can be more complex and time-consuming.
- C++ offers libraries such as Poppler and libHaru, which effectively process PDF documents. However, the complexity of C++ development makes it less convenient for tasks related to image processing.

When developing applications for converting images to PDF, C# together with the iTextSharp library is one of the most productive and convenient options. Ease of integration, support for many image formats and high performance make this solution optimal for such tasks.

An application developed in C# using the iTextSharp library is designed to convert images to PDF format. The main goal of the program is to provide the user with a convenient tool for automatically creating a PDF document from one or more images. This solution can find application in such areas as archiving graphic data, creating digital documents or preparing reports.

The program includes several main modules:

1. Image loading: The user can select one or more images in various formats (e.g. JPEG, PNG, BMP) that will be converted to PDF.
2. Image scaling: When adding images to PDF, the program automatically adjusts them to the page size, ensuring correct display without distortion or loss of quality.
3. PDF creation: Using the iTextSharp library, the program generates a PDF document by adding images to individual pages or composing them according to the user's settings.
4. PDF saving: Once the conversion is complete, the program prompts the user to save the finished PDF document to the device.
5. This program uses the iTextSharp library to create and edit PDF documents. It provides convenient tools for working with text and graphic elements within PDF files. The ScaleToFit method is used to automatically adjust the image to the page size, which helps avoid distortion or cropping of the image. Each image is placed on a separate page using the NewPage method, providing a convenient document structure in which each image is displayed on its own page. After adding all the images, the PDF document is saved to the location specified by the user.
6. The program has a simple and intuitive interface that simplifies the process of converting images to PDF without the need for complex settings. It supports various image formats and automatic scaling, which ensures their correct display in PDF. Thanks to the use of C# and iTextSharp, the program efficiently processes a large number of images in a short time, which makes it especially useful for working with large data archives. This utility can be

used in various areas, such as document automation or combining images into PDF for presentations and demonstrations.

7. The algorithm for scaling and embedding images in a PDF document in C# using the iTextSharp library consists of several basic steps. This process ensures the correct processing of images of different sizes, adapting them to the parameters of the PDF pages and preventing distortion or loss of quality. At the initial stage, the program loads images for embedding in a PDF document, supporting various formats (for example, JPEG, PNG, BMP), and checks whether they are loaded correctly and in the correct format.

8. After that, the program creates a new PDF document with the specified page parameters, such as size and orientation. The page size can be selected based on the image parameters or specified by the user. For each image, the program calculates its width and height, which is necessary to determine whether the image needs to be scaled so that it is correctly displayed in the document, without cropping or distortion. If the image dimensions exceed the page dimensions, the program performs a scaling operation, calculating a coefficient depending on the ratio of the image and page sizes.

After scaling, the image is integrated into the PDF document. To improve visual perception, the program places the image in the center of the page, using the alignment parameters. This allows you to neatly arrange the image horizontally and vertically. Then the image is added to the current page of the PDF document. To prevent images from overlapping, each image is placed on a separate page using the NewPage method. If the user selects multiple images, a separate page is created for each. The scaling and embedding process is repeated for all images. Once the insertion is complete, the program completes the creation of the PDF document and saves it in the directory selected by the user[4,8].

Optimizing an image to PDF converter program in C# involves several key steps to improve performance, reduce resource usage, and ensure stable operation when processing large amounts of data. This section covers the main methods that can help improve the efficiency of the program. One of the most important factors is effective memory management, especially when working with large images. Uncontrolled loading and processing can lead to delays or even failures due to lack of resources. The following approaches can be used to solve this problem:

- Processing large images in parts reduces the memory load by dividing the image into fragments for sequential processing.
- Setting a limit on the number of simultaneously processed images prevents memory overflow when working with several large files.
- It is important to release memory in a timely manner after finishing processing the image. In C#, this can be done using the Dispose() method, which releases occupied resources.

Example of freeing resources:

```
using (Image img = Image.GetInstance(imagePath))
{
img.ScaleToFit(pageWidth, pageHeight);
img.Alignment = Image.ALIGN_CENTER;
pdfDoc.Add(img);
pdfDoc.NewPage();
}
```

The size of the resulting PDF document is also an important aspect of optimization. Embedding large images without prior compression can significantly increase its size. To minimize the file size, it is recommended to compress images before adding them to the PDF. Compression can be done by saving images as low-quality JPEGs, which reduces their size with minimal loss of visual quality.

Example of compression:

```
Image img = Image.GetInstance(imagePath);
img.CompressionLevel = 9;
```

Using data buffering allows you to avoid unnecessary writes to disk, which speeds up streaming operations. This is especially important when working with large PDF files and many images. Buffering helps speed up PDF creation and minimize latency. Using streaming writing via MemoryStream can significantly improve the program's speed.

Example of using a buffer:

```
using (MemoryStream ms = new MemoryStream())
{
PdfWriter.GetInstance(pdfDoc, ms);
File.WriteAllBytes(outputPdfPath, ms.ToArray());
}
```

Scaling images to optimal sizes before inserting them into a PDF reduces memory load and speeds up the program. Since many images have a resolution larger than required for PDF pages, pre-scaling them to the required size reduces the file size and speeds up processing.

Scaling example:

```
img.ScaleToFit(pageWidth, pageHeight);
```

To increase performance when working with a large number of images, you can use multithreading. Parallel data processing allows you to distribute tasks between processor cores, speeding up program execution on multi-core systems. This can be done using Task or Parallel.ForEach.

An example of using multithreading:

```
Task.Run(() =>
{
foreach (var imagePath in imagePaths)
{
// Process each image
}
});
```

Automatic memory management in C# via the Garbage Collector effectively manages program resources. In certain cases, you can force the garbage collector to free up memory after processing each image.

An example of calling the garbage collector:

```
GC.Collect();
GC.WaitForPendingFinalizers();
```

Thus, the use of these optimization methods will significantly improve the program's performance when working with large volumes of images and PDF documents, making it more stable and efficient.

The program for converting images to PDF in C# is used in various fields where automation of graphic data processing and management processes is required. Let's consider several examples of its use in real-life scenarios.

One of the key areas of application of the program is the digitization of paper documents. Many companies and archives are faced with the need to convert documents into a digital format for their convenient storage and search. The program allows you to convert scans of paper documents into PDF files, simplifying the archiving process and making documents available for searching and viewing. For example, an archival institution can use the program to combine hundreds of scanned documents into single PDF files for further use.

The program is also suitable for creating electronic photo albums and presentations in which images are automatically converted to PDF documents. This format is convenient for storage, printing and transmission, since PDF files often take up less space than original images and are easy to open on any device. For example, photo studios can use the program to quickly create photo albums from client images, which can then be emailed or printed.

Medical institutions can use the program to combine the results of diagnostic tests, such as X-rays or MRIs, into a single PDF document. This simplifies the storage of medical data and its transfer between specialists. For example, a clinic can use the program to automatically convert patient images into PDF documents for further analysis and storage in an electronic database.

In companies, the program can be used to prepare reports with graphical elements such as images, graphs, and charts. It automates the process of creating reports that include illustrations and graphical data in a single document. For example, a marketing department can use the program to automatically integrate images of graphs with text information into a single PDF report for presentation to management.

For designers, photographers, and artists, the program can be a convenient tool for creating a portfolio. It allows you to automatically convert images to PDF, eliminating the need to manually embed images into a document. For example, a designer can collect all of their work into a single PDF document and send it to potential clients or employers.

In addition, the program can be useful for creating training materials or technical documentation, especially when you need to work with a large number of screenshots. It automatically converts screenshots to PDF, simplifying the creation of instructions or manuals. For example, a teacher can use the program to create a PDF document from screenshots of their training materials, which can then be shared with students.

CONCLUSIONS

Developed in C#, the program for converting images to PDF format is a universal solution that can automate work with graphic data. Due to its flexible capabilities and wide range of applications, this program is in demand in such areas as document digitization, creation of electronic albums, processing of medical images, and automation of various reporting processes. The main advantage of the program is its functionality for scaling images, proper integration into a PDF document and optimizing files to achieve a balance between quality and size. This makes the program ideal for companies and organizations where it is important to reduce the time and effort spent on processing graphic files.

Additional optimization of the program, including methods such as image compression and memory management, allows you to effectively work with large amounts of data. The program can be easily integrated into larger systems and is used on various platforms, which expands its functionality and makes it suitable for different types of applications - from desktop to web services.

Thus, the program simplifies tasks related to image processing, improves document flow processes and provides a practical solution for many areas. Prospects for its development include adding new features, improving performance and supporting more formats, which makes it even more useful and in demand among users.

References:

1. Shoykulov Sh.K. (2024). USING PYTHON PROGRAMMING IN COMPUTER GRAPHICS. https://doi.org/10.5281/zenodo.13926022
2. Shoyqulov, S. (2024). DATA VISUALIZATION IN PYTHON. В EURASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES (Т. 4, Выпуск 10, сс. 15–22). Zenodo. https://doi.org/10.5281/zenodo.13892777
3. Shoyqulov, S. (2024). GRAPHICAL PROGRAMMING OF 2D APPLICATIONS IN C#. В EURASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES (Т. 4, Выпуск 10, сс. 7–14). Zenodo. https://doi.org/10.5281/zenodo.13892766
4. Bozorov, A., & Shoyqulov, S. (2024). COMPUTER GRAPHICS IN TECHNICAL DISCIPLINES. В EURASIAN JOURNAL OF ACADEMIC RESEARCH (Т. 4, Выпуск 10, сс. 21–27). Zenodo. https://doi.org/10.5281/zenodo.13898180
5. Bozorov, A., & Shoyqulov, S. (2024). COMPUTER GRAPHICS IN THE NATURAL SCIENCES. В EURASIAN JOURNAL OF ACADEMIC RESEARCH (Т. 4, Выпуск 10, сс. 12–20). Zenodo. https://doi.org/10.5281/zenodo.13898146
6. Shoyqulov Sh. Q.METHODS FOR PLOTTING FUNCTION GRAPHS IN COMPUTERS USING BACKEND AND FRONTEND INTERNET TECHNOLOGIES. European Scholar Journal (ESJ). Vol. 2 No. 6, June 2021, ISSN: 2660-5562. P.161-165, https://scholarzest.com/index.php/esj/article/view/964/826
7. Sh.Q. Shoyqulov. (2021). Methods for plotting function graphs in computers using backend and frontend internet technologies. European Scholar Journal, 2(6), 161-165. Retrieved from https://scholarzest.com/index.php/esj/article/view/964
8. Sh.Q. Shoyqulov. (2022). The text is of the main components of multimedia technologies. Academicia Globe: Inderscience Research, 3(04), 573–580. https://doi.org/10.17605/OSF.IO/VBY8Z