



Generate ASCII Art – A Simple How To in C#



Dev Leader

28 Aug 2023 CPOL

Change a picture into ASCII art using your own C# program

Have you ever wanted to change a picture into ASCII art? Now you can with your very own C# program that can generate ASCII art! Sample code is included!



ASCII art, a technique of creating visuals using characters from the ASCII standard, has been a part of the computing world for decades. It's a fascinating way to represent images without the need for traditional graphics. For those new to programming, building a program to generate ASCII art can serve as an insightful introduction.

In this guide, we'll walk through a C# approach to transform standard images and generate ASCII art from them. Not only will you have the full source code to have a functioning C# app that can generate

ASCII art, but I'll also explain why simple programs like this can be critical for helping you learn.

Before I Provide Code to Generate ASCII Art

I realize many of you coming here are just looking to jump directly into the code. I get it! But that's why I want to put an important message beforehand, especially for the more junior developers.

Many times, beginner programmers are stuck in some of the early phases of learning because they are not sure how to allocate their time. They are trying to read books, articles, and blog posts (just like this one!) to learn theory, or watching videos and trying to find the best BootCamp so they have the best chance of success. I regularly remind my audience that I think building things and actually writing code is one of the absolute best ways to learn.

As we navigate this code together, I want you to keep this in mind! At the end of the article, I propose some variations and enhancements that you may want to consider. I've included this list not just because I think it's pretty cool, but to get your creative juices flowing! Think about the different things you want to focus on as a developer and see if you can incorporate them into your ASCII art generator!

Being able to leverage simple programs like this takes the stress away from "what's the right thing to build" and allows you to focus on learning and exploring. Watch the video as you follow along!

Make Your Art UNIQUE - Build Your Own ASCII Art Generator in #dotnet



Example Code to Generate ASCII Art

Alright, you toughed it out through my introduction. Thanks! Let's look at some code (which, by the way, is available in full on GitHub):

C#

```
string imagePath = "your file path here";

using var inputStream = new FileStream(
    imagePath,
    FileMode.Open,
    FileAccess.Read,
    FileShare.Read);
var generator = new Generator();

using var sourceImage = Image.Load(inputStream);
using var imageRgba32 = sourceImage.CloneAs<Rgba32>();
using var image = new ImageSharpImageSource(imageRgba32);

var asciiArt = generator.GenerateAsciiArtFromImage(image);

Console.WriteLine(asciiArt.Art);
```

This is the entry point to our C# program. Here, we're setting up the path to our image and creating a stream to read it. We also instantiate our main Generator class, which will handle the ASCII conversion, along with the ImageSharpImageSource that will hold the image data. The magic happens inside of the GenerateAsciiArtFromImage method, which we will look at shortly.

The ImageSharp library is used to load the image and then clone it into a format (Rgba32) that allows us to work with individual pixel colors. The ImageSharpImageSource class acts as a bridge between the ImageSharp library and our ASCII generation logic. When we look at the code for this class, we'll be able to see the indexer method that allows us to get the pixel data for an X and Y coordinate.

Let's look at the implementation for the image source next:

C#

```
internal interface IImageSource : IDisposable
{
    int Width { get; }

    int Height { get; }

    float AspectRatio { get; }

    Rgb GetPixel(int x, int y);
}

internal sealed class ImageSharpImageSource : IImageSource
{
    private readonly Image<Rgba32> _image;

    public ImageSharpImageSource(Image<Rgba32> image)
    {
        _image = image;
    }

    public int Width => _image.Width;

    public int Height => _image.Height;

    public float AspectRatio => _image.Width / (float)_image.Height;

    public Rgb GetPixel(int x, int y)
    {
        var pixel = _image[x, y];
        return new(
            pixel.R,
            pixel.G,
            pixel.B);
    }

    public void Dispose() => _image.Dispose();
}
```

In the above code, we can see that we are implementing the `IImageSource` interface. This was done because you can actually implement this same functionality with the `System.Drawing` namespace and the `Bitmap` class, but it will only work on Windows. The code `_image[x, y]` allows us to get the pixel information from the image!

The final class of importance is the actual generator. We'll examine the code in more detail in the following sections:

C#

```
internal sealed class Generator
{
    public AsciiArt GenerateAsciiArtFromImage(
        IImageSource image)
    {
        var asciiChars = "@%#*+=-:,. ";

        var aspect = image.Width / (double)image.Height;
        var outputWidth = image.Width / 16;
        var widthStep = image.Width / outputWidth;
        var outputHeight = (int)(outputWidth / aspect);
        var heightStep = image.Height / outputHeight;

        StringBuilder asciiBuilder = new(outputWidth * outputHeight);
        for (var h = 0; h < image.Height; h += heightStep)
        {
            for (var w = 0; w < image.Width; w += widthStep)
            {
                var pixelColor = image.GetPixel(w, h);
                var grayValue = (int)(pixelColor.Red * 0.3 +
                    pixelColor.Green * 0.59 + pixelColor.Blue * 0.11);
                var asciiChar = asciiChars[grayValue * (asciiChars.Length - 1) / 255];
                asciiBuilder.Append(asciiChar);
                asciiBuilder.Append(asciiChar);
            }

            asciiBuilder.AppendLine();
        }

        AsciiArt art = new(
            asciiBuilder.ToString(),
            outputWidth,
            outputHeight);
        return art;
    }
}
```

Breaking Down the Image Processing

When we talk about images in computers, we're essentially discussing a matrix of pixels. Each pixel has a color, and this color is typically represented by three primary components: Red, Green, and Blue

(RGB). You might also see a 4th component in this mix, which is “alpha” (or transparency) represented by an A (RGBA). The combination of these components in varying intensities gives us the vast spectrum of colors we see in digital images.

ASCII art doesn't deal with colors in the traditional sense. Instead, it represents images using characters that have varying visual weights or densities. This is where the concept of grayscale comes into play. A grayscale image is one where the RGB components of each pixel have the same value, resulting in various shades of gray. The significance of converting an image to grayscale for ASCII art is to simplify the representation. By reducing an image to its luminance, we can then map different shades of gray to specific ASCII characters and generate ASCII art from an image.

In our code, the `IImageSource` interface serves as an abstraction for our image source. It provides properties to get the width, height, and aspect ratio of the image and a method to retrieve the color of a specific pixel. The `ImageSharpImageSource` class is an implementation of this interface using the `ImageSharp` library. As we saw, it wraps around an `ImageSharp` image and provides the necessary data for our program to generate ASCII art.

As we'll see in a later section, there are still some considerations around image scaling including downsizing the image to fit into the console output and considering aspect ratios. Additionally, the code itself has not been benchmarked to see if there are opportunities to reduce memory usage and/or generate the output more effectively.

The Generator Class: The Key To Generate ASCII Art

The `Generator` class is where the magic happens. It's responsible for transforming our image into a piece of ASCII art. Let's dive deeper into its primary method: `GenerateAsciiArtFromImage`.

C#

```
var asciiChars = "@%#*+==-.:,. ";
```

This line defines our palette of ASCII characters. These characters are chosen based on their visual density, with `@` being the densest and a space () being the least dense. You can customize this list to have different visual appearances and add more or remove some characters to change the granularity of the shading being used.

C#

```
var aspect = image.Width / (double)image.Height;  
var outputWidth = image.Width / 16;  
var widthStep = image.Width / outputWidth;  
var outputHeight = (int)(outputWidth / aspect);  
var heightStep = image.Height / outputHeight;
```

This code is actually incomplete, but it's a good opportunity to think about enhancements. The purpose of this block is to work on getting the right output resolution of the image and considering

how it needs to be scaled. It would be ideal to have this be configurable so there are no magic numbers!

One important detail that we have for looping through each pixel in the image is that we start from the top left and then work across the row before going to the next row. This is because it's much more straightforward to print a line to the console than to print column by column. As we loop through the image's pixels, we need to determine which ASCII character best represents a particular pixel's color. To do this, we first convert the pixel's color to a grayscale value:

C#

```
var pixelColor = image.GetPixel(w, h);
var grayValue = (int)(pixelColor.Red * 0.3 +
                    pixelColor.Green * 0.59 + pixelColor.Blue * 0.11);
```

This formula can be tweaked to get a grayscale value, but the current magic numbers here emphasize the green component due to the human eye's sensitivity to it. With the grayscale value in hand, we map it to one of our ASCII characters:

C#

```
var asciiChar = asciiChars[grayValue * (asciiChars.Length - 1) / 255];
```

This mapping ensures that darker pixels get represented by denser ASCII characters and lighter pixels by less dense characters. The resulting character is then added to our ASCII art representation.

Going Cross Platform: ImageSharp to Generate ASCII Art

ImageSharp is a powerful, open-source library in the .NET ecosystem that provides functionality for processing images. It's versatile, efficient, and supports a wide range of image formats. To generate ASCII art, we need a way to read and manipulate images, and ImageSharp fits the bill perfectly.

Because I started this project initially using `System.Drawing`, I wanted to illustrate that we could interchangeably use ImageSharp or `System.Drawing` to generate ASCII art. By doing this, I could pull the core logic into one spot, and abstract access to the more image-specific code. And why do this? Well, what if we decide to switch to another image processing library in the future? To keep things modular and maintainable, we introduce an abstraction layer: the `IImageSource` interface.

The `ImageSharpImageSource` class serves as a bridge between ImageSharp and our generator. It implements the `IImageSource` interface, wrapping around an ImageSharp image.

C#

```
public Rgb GetPixel(int x, int y)
{
    var pixel = _image[x, y];
```

```
    return new( pixel.R, pixel.G, pixel.B);  
}
```

The `GetPixel` method is particularly significant. It retrieves the RGB values of a specific pixel in the image. This method is crucial for our generator, as it uses these RGB values to determine the grayscale value and, subsequently, the appropriate ASCII character for that pixel. Any image library we want to use would then need some way for us to access a specific pixel, which seems like a reasonable feature to expect.

Generate ASCII Art: Project Enhancements

To generate ASCII art, we came up with a solid foundation, but there's always room for improvement and innovation. To re-iterate, projects like this are awesome for learning because the direction you head in with your enhancements can let you pick and choose how to focus your energy on learning. Here are some suggestions to take this project to the next level:

1. Refining ASCII Art Generation

Different images can benefit from different sets of ASCII characters. Experiment with various character sets to see which ones produce the best results for different kinds of images. Could this be something that you allow the user to configure?

What about the image size and scaling? Can we fit the output to specific dimensions? What about maintaining the aspect ratio or not? These are extras that could really enhance the usability of the generator!

2. Adding Color

While traditional ASCII art is monochrome, there's no reason you can't introduce color. By mapping pixel colors to terminal or HTML color codes, you can produce vibrant, colored ASCII art. Even in our C# console on Windows, we do have some basic colors we can work with, so before completely overhauling what's here you could alter the code to generate ASCII art such that it returns the pixel color, and have something that can map that to the closest console color!

3. Performance Optimization

For larger images, the generation process could be a bit slow. Dive into performance profiling and see if there are bottlenecks you can address, perhaps by optimizing loops or leveraging parallel processing. You might even be able to use something more effective than a `StringBuilder` here... Even if we go back to basics on some of the C# data types we could perhaps pick something that is more performant!

4. Web Application or GUI Integration

Turn this console-based tool into a more user-friendly application. Imagine a web application where users can upload images and instantly see their ASCII art representation. Or a desktop application with a GUI that allows users to tweak settings in real time. You have so many options to explore just by moving away from a traditional console application!

Conclusion

As it turns out, characters in the range from 0 to 255 can teach us a whole lot when we channel that into a program that can generate ASCII art! While the project might seem straightforward on the surface, it offers a deep dive into various programming concepts, from image processing to algorithmic optimization. With some optional enhancements, you can really spice things up and learn some awesome new things!

For those who've followed along, I hope you've gained not just the knowledge of how to create ASCII art, but also an appreciation for the learning opportunities that even simple projects can present. The real value lies not just in the end product but in the process of problem-solving, experimenting, and iterating.

I encourage you to take this code, adapt it, and make it your own. Experiment with different features, optimize the code further, or integrate it into larger projects. Remember, every line of code you write, and every challenge you overcome, adds to your growth as a developer. And sometimes, it's the simplest projects that offer the most profound lessons!

License

This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOLO)

Written By

Dev Leader

Team Leader Microsoft

 United States

I'm a software engineering professional with a decade of hands-on experience creating software and managing engineering teams. I graduated from the University of Waterloo in Honours Computer Engineering in 2012.

I started blogging at <http://www.devleader.ca> in order to share my experiences about leadership (especially in a startup environment) and development experience. Since then, I have been trying to create content on various platforms to be able to share information about programming and engineering leadership.

My Social:

YouTube: <https://youtube.com/@DevLeader>

TikTok: <https://www.tiktok.com/@devleader>

Blog: <http://www.devleader.ca/>

GitHub: <https://github.com/ncosentino/>

Twitch: <https://www.twitch.tv/ncosentino>

Twitter: <https://twitter.com/DevLeaderCa>

Facebook: <https://www.facebook.com/DevLeaderCa>


Instagram:

<https://www.instagram.com/dev.leader>

LinkedIn: <https://www.linkedin.com/in/nickcosentino>



Comments and Discussions

 **2 messages** have been posted for this article Visit

<https://www.codeproject.com/Articles/5367237/Generate-ASCII-Art-A-Simple-How-To-in-Csharp> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#)

[Advertise](#)

[Privacy](#)

[Cookies](#)

[Terms of Use](#)

Article Copyright 2023 by Dev Leader
Everything else Copyright © CodeProject,
1999-2023

Web04 2.8:2023-08-14:1