
PureImage

PureImage is a pure 100% JavaScript implementation of the HTML Canvas 2D drawing API for NodeJS. It has no native dependencies. You can use it to resize images, draw text, render badges, convert to grayscale, or anything else you could do with the standard Canvas 2D API. It also has additional APIs to save an image as PNG and JPEG.

Typescript Rewrite

As of version 0.4.* PureImage has been rewritten in 100% Typescript. The module is compiled to both Common JS and ES Modules. If it was working for you before it should still work, but if you notice anything off please file a bug report.

Also *note* that `font.load()` now returns a promise instead of using a callback. If you need synchronous support you can still fuse `font.loadSync()`.

Installation

```
1 npm install pureimage
```

Simple example

Make a 100x100 image, fill with red, write to png file

```
1 import * as PImage from "pureimage";
2 import * as fs from "fs";
3
4 // make image
5 const img1 = PImage.make(100, 100);
6
7 // get canvas context
8 const ctx = img1.getContext("2d");
9
10 // fill with red
11 ctx.fillStyle = "red";
12 ctx.fillRect(0, 0, 100, 100);
13
14 //write to 'out.png'
15 PImage.encodePNGToStream(img1, fs.createWriteStream("out.png"))
16   .then(() => {
```

```
17     console.log("wrote out the png file to out.png");
18   })
19   .catch((e) => {
20     console.log("there was an error writing");
21   });
```

result



supported Canvas Features

note: PureImage values portability and simplicity of implementation over speed. If you need maximum performance you should use a different library backed by native code, such as Node-Canvas

- set pixels
- stroke and fill paths (rectangles, lines, quadratic curves, bezier curves, arcs/circles)
- copy and scale images (nearest neighbor)
- import and export JPG and PNG from streams using promises
- render basic text (no bold or italics yet)
- anti-aliased strokes and fills
- transforms
- standard globalAlpha and rgba() alpha compositing
- clip shapes

On the roadmap, but still missing

- gradients fills
- image fills
- blend modes besides SRC OVER
- smooth clip shapes
- bold/italic fonts
- smooth image interpolation

Why?

There are more than enough drawing APIs out there. Why do we need another? My personal hatred of C/C++ compilers is widely known. The popular Node module Canvas.js does a great job, but it's backed by Cairo, a C/C++ layer. I hate having native dependencies in Node modules. They often don't compile, or break after a system update. They often don't support non-X86 architectures (like the Raspberry Pi). You have to have a compiler already installed to use them, along with any other native dependencies pre-installed (like Cairo).

So, I made PureImage. Its goal is to implement the HTML Canvas spec in a headless Node buffer. No browser or window required.

PureImage is meant to be a small and maintainable Canvas library. It is *not meant to be fast*. If there are two choices of algorithm we will take the one with the simplest implementation, and preferably the fewest lines. We avoid special cases and optimizations to keep the code simple and maintainable. It should run everywhere and be always produce the same output. But it will not be fast. If you need speed go use something else.

PureImage uses only pure JS dependencies. OpenType for font parsing, PngJS for PNG import/export, and jpeg-js for JPG import/export.

Examples

Make a new empty image, 100px by 50px. Automatically filled with 100% opaque black.

```
1 var PImage = require("pureimage");
2 var img1 = PImage.make(100, 50);
```

Fill with a red rectangle with 50% opacity

```
1 var ctx = img1.getContext("2d");
2 ctx.fillStyle = "rgba(255,0,0, 0.5)";
3 ctx.fillRect(0, 0, 100, 100);
```

Fill a green circle with a radius of 40 pixels in the middle of a 100px square black image.

```
1 var img = PImage.make(100, 100);
2 var ctx = img.getContext("2d");
3 ctx.fillStyle = "#00ff00";
4 ctx.beginPath();
5 ctx.arc(50, 50, 40, 0, Math.PI * 2, true); // Outer circle
6 ctx.closePath();
7 ctx.fill();
```



Draw the string 'ABC' in white in the font 'Source Sans Pro', loaded from disk, at a size of 48 points.

```
1 test("font test", (t) => {
2   var fnt = PImage.registerFont(
3     "test/fonts/SourceSansPro-Regular.ttf",
4     "Source Sans Pro",
5   );
6   fnt.loadSync();
7   var img = PImage.make(200, 200);
8   var ctx = img.getContext("2d");
9   ctx.fillStyle = "#ffffff";
10  ctx.font = "48pt 'Source Sans Pro'";
11  ctx.fillText("ABC", 80, 80);
12 });
```

Write out to a PNG file

```
1 PImage.encodePNGToStream(img1, fs.createWriteStream("out.png"))
2   .then(() => {
3     console.log("wrote out the png file to out.png");
4   })
5   .catch((e) => {
6     console.log("there was an error writing");
7   });
```

Read a jpeg, resize it, then save it out

```
1 PImage.decodeJPEGFromStream(fs.createReadStream("test/images/bird.jpg"))
2   .then(
3     (img) => {
4       console.log("size is", img.width, img.height);
5       var img2 = PImage.make(50, 50);
6       var c = img2.getContext("2d");
7       c.drawImage(
8         img,
9         0,
10        0,
11        img.width,
12        img.height, // source dimensions
13        0,
14        0,
15        50,
16        50);
```

```
15     50, // destination dimensions
16   );
17   var pth = path.join(BUILD_DIR, "resized_bird.jpg");
18   PImage.encodeJPEGToStream(img2, fs.createWriteStream(pth), 50).then
      (() => {
19     console.log("done writing");
20   });
21 },
22 );
```

This examples streams an image from a URL to a memory buffer, draws the current date in big black letters, and writes the final image to disk

```
1  import * as PImage from "pureimage";
2  import fs from "fs";
3  import * as client from "https";
4  let url =
5    "https://vr.josh.earth/webxr-experiments/physics/jinglesmash.
      thumbnail.png";
6  let filepath = "output_stream_sync.png";
7  //register font
8  const font = PImage.registerFont(
9    "../test/unit/fixtures/fonts/SourceSansPro-Regular.ttf",
10   "MyFont",
11  );
12  //load font
13  font.loadSync();
14  //get image
15  client.get(url, (image_stream) => {
16    //decode image
17    PImage.decodePNGFromStream(image_stream).then((img) => {
18      //get context
19      const ctx = img.getContext("2d");
20      ctx.fillStyle = "#000000";
21      ctx.font = "60pt MyFont";
22      ctx.fillText(new Date().toLocaleDateString(), 50, 80);
23      PImage.encodePNGToStream(img, fs.createWriteStream(filepath)).then
        (() => {
24        console.log("done writing to ", filepath);
25      });
26    });
27  });
```

produces



The same as above but with Promises

```
1 import * as PImage from "pureimage";
2 import fs from "fs";
3 import * as client from "https";
4
5 let url =
6   "https://vr.josh.earth/webxr-experiments/physics/jinglesmash.
   thumbnail.png";
7 let filepath = "output.png";
8 let fontpath = "test/unit/fixtures/fonts/SourceSansPro-Regular.ttf";
9 PImage.registerFont(fontpath, "MyFont")
```

```

10 .loadPromise()
11 //Promise hack because https module doesn't support promises natively
12 )
13 .then(() => new Promise((res) => client.get(url, res)))
14 .then((stream) => PImage.decodePNGFromStream(stream))
15 .then((img) => {
16     //get context
17     const ctx = img.getContext("2d");
18     ctx.fillStyle = "#000000";
19     ctx.font = "60pt MyFont";
20     ctx.fillText(new Date().toLocaleDateString(), 50, 80);
21     return PImage.encodePNGToStream(img, fs.createWriteStream(filepath)
22     );
23 })
24 .then(() => {
25     console.log("done writing", filepath);
26 });

```

The same as above but with async await

```

1 import fs from "fs";
2 import * as https from "https";
3 const https_get_P = (url) => new Promise((res) => https.get(url, res));
4
5 async function doit() {
6     let url =
7         "https://vr.josh.earth/webxr-experiments/physics/jinglesmash.
8         thumbnail.png";
9     let filepath = "output_stream_async.png";
10    //register font
11    const font = PImage.registerFont(
12        "../test/unit/fixtures/fonts/SourceSansPro-Regular.ttf",
13        "MyFont",
14    );
15    //load font
16    await font.load();
17    //get image
18    let image_stream = await https_get_P(url);
19    //decode image
20    let img = await PImage.decodePNGFromStream(image_stream);
21    //get context
22    const ctx = img.getContext("2d");
23    ctx.fillStyle = "#000000";
24    ctx.font = "60pt MyFont";
25    ctx.fillText(new Date().toLocaleDateString(), 50, 80);
26    await PImage.encodePNGToStream(img, fs.createWriteStream(filepath));
27    console.log("done writing to ", filepath);
28 }
29 doit()
30 .then(() => console.log("done"))
31 .catch((e) => console.error(e));

```

Save a canvas to a NodeJS buffer as PNG using a `PassThrough` stream:

```
1 import * as PImage from "pureimage";
2 import { PassThrough } from "stream";
3 const passThroughStream = new PassThrough();
4 const pngData = [];
5 passThroughStream.on("data", (chunk) => pngData.push(chunk));
6 passThroughStream.on("end", () => {});
7 pureimage.encodePNGToStream(canvas, passThroughStream).then(() => {
8   let buf = Buffer.concat(pngData);
9   expect(buf[0]).to.eq(0x89);
10  expect(buf[1]).to.eq(0x50);
11  expect(buf[2]).to.eq(0x4e);
12  expect(buf[3]).to.eq(0x47);
13  done();
14 });
```

Troubleshooting

missing or broken text

PureImage uses OpenType.js to parse fonts and rasterize glyphs. If you are having trouble rendering something first check on the OpenType website that the font can actually be parsed and rendered. If you are rendering non-latin character sets you may need to install an additional dependency to your operating system. For example, rendering arabic text may require `pip install arabic-reshaper` on Linux.

Using a really large image buffer

PureImage has no inherit size limitations, but NodeJS does have a default max memory setting. You can learn how to increase the default [here](#)

New 0.4.x release

After another long lull, I've ported PureImage to Typescript. Most of the work was actually done by the amazing and talented Josh Hemphill. As part of this port I switched to using esbuild for compiling & packaging the Typescript, and Vitest for unit tests. They are vastly faster than our old system.

This release also fixes tons of bugs and adds some small features:

- updated PngJS, OpenType.js and JPegJS to their latest version.

-
- Node 14 is now the minimum supported version
 - linear and radial gradient fills are supported. See `test/gradientfill.test.ts`)

New 0.3.x release

After a long lull, I've ported the code to modern ES6 modules, so you can just do an `import pureimage from 'pureimage'` like any other proper modern module. If you are using `require('pureimage')` it should just work thanks to the `dist/pureimage-umd.cjs` file built with Rollup. It also has a stub to let `pureimage` run in the browser and delegate to the real HTML canvas. This helps with isomorphic apps.

Other updates include

- Switch to MochaJS for the unit tests.
- add more unit tests.
- support drawing images when using transforms
- implement `rect()`
- implement `ImageData` with `getImageData()` and `putImageData()`
- fix gradient fill
- add all CSS named colors
- support `#rgb`, `#rgba`, and `#rrggbbbaa` color strings
- applied more bug fixes from PRs, thanks to our contributors.

New 0.1.x release

I've completely refactored the code so that it should be easier to maintain and implement new features. For the most part there are no API changes (since the API is defined by the HTML Canvas spec), but if you were using the font or image loading extensions you will need to use the new function names and switch to promises. For more information, please see the API docs

I'm also using Node buffers instead of arrays internally, so you can work with large images faster than before. Rich text is no longer supported, which is fine because it never really worked anyway. We'll have to find a different way to do it.

I've tried to maintain all of the patches that have been sent in, but if you contributed a patch please check that it still works. Thank you all! - josh

Thanks!

Thanks to Nodebox / EMRG for opentype.js

Thanks to Rosetta Code for Bresenham's in JS

Thanks to Kuba Niegowski for PngJS

Thanks to Eugene Ware for jpeg-js

Thanks for patches from:

- Dan danielbarela
- Eugene Kulabuhov ekulabuhov
- Lethexa lethexa
- The Louie the-louie
- Jan Marsch kekscom