Image Processing and Pixel Art

Derek Otcasek
Derek.Otcasek001@umb.edu
University of Massachusetts Boston

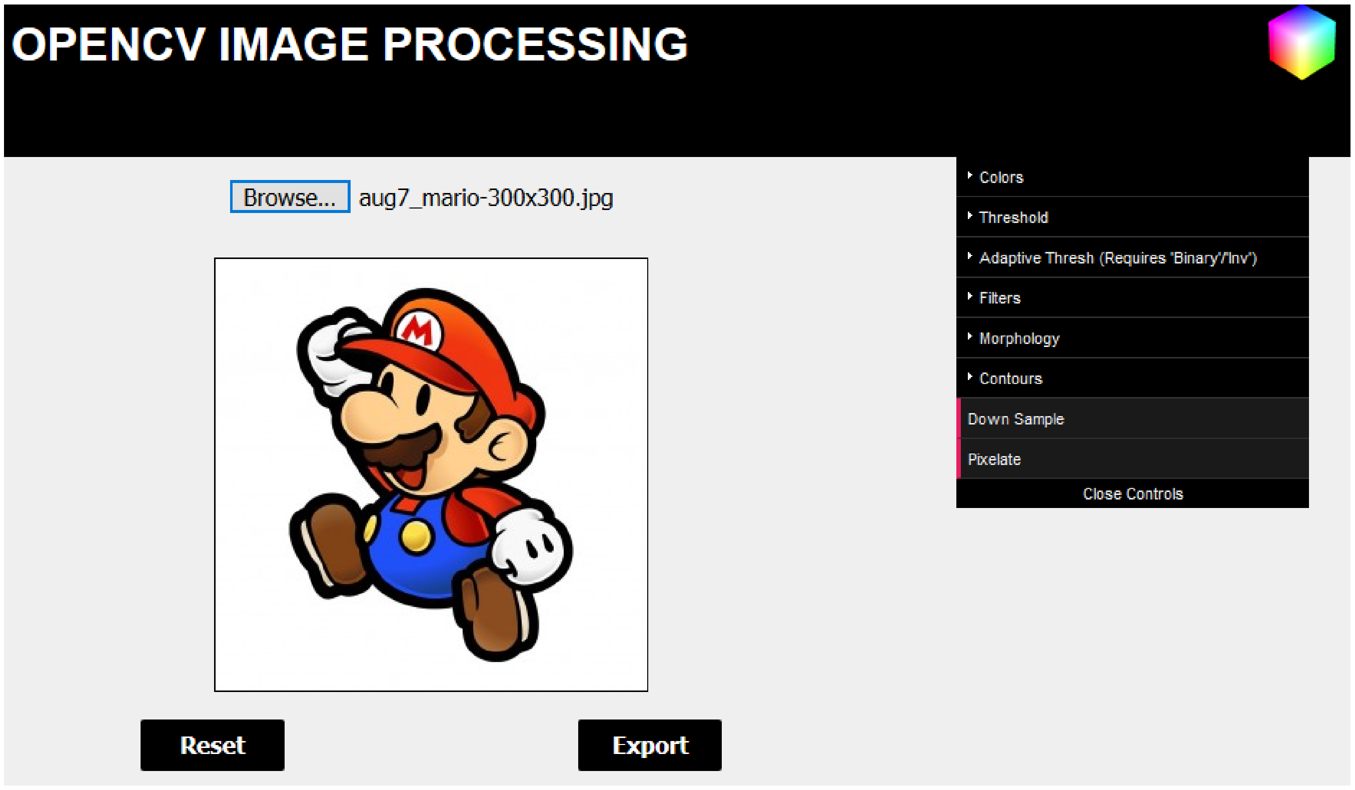

Figure 1: A screen shot of the interface.

ABSTRACT

This project allows a user to be able to upload an image of their choice and perform image processing operations on the image in real time using a Dat.GUI sidebar. They are given the option to save the result to a file. The user also has the option to display the resulting image as pixel art which is rendered using 3d cubes with XTK.

KEYWORDS

WebGL, Visualization

ACM Reference Format:
Derek Otcasek. 2019. Image Processing and Pixel Art. In *CS460: Computer Graphics at UMass Boston, Fall 2019.* Boston, MA, USA, 3 pages. https://CS460.org

1 INTRODUCTION

Being able to perform image processing in real time is helpful for learning about the different types of image transformations and commonly used techniques. By incorporating the pixel art it makes it fun and easy to see the effects of interpolation.

2 RELATED WORK

This project uses XTK [2], Three.js [1], and OpenCV [3].

3 METHOD

When you come to the initial landing page, a test image is preloaded. there is a button which allows you to input a image file, as well as a button to reset the image to the original and a button to export. There is a Dat.GUI sidebar with controls to process the image. The Dat.GUI implements controls for different categories.

- Colors: This lets you select between gray-scale, HSV, and single channel.
- Threshold: This lets you select different types of simple threshold values as well as has sliders for the associated parameters.
- Adaptive Thresholding: This lets you select thresholds based on neighboring pixels instead of each pixel individually.
- Filters: This lets you select different Blur effects
- Morphology: This lets you select different morphological effects as well as allows for the selection of iterations.
- Contours: Draws the contours in a random color. sliders are added to filter out contours based on the area.
- Down Sample: reduces the sample size if the image is too large to process.
- Pixelate: renders the image as a 3D interpretation of the image as pixel art.

3.1 Implementation

I was able to implement these functions in real time by adding change listeners to the sliders and updating the image when parameters are changed. For each folder in Dat.GUI i usually wrote a class which contained default values for the variables which could be potentially changed.

```
var Thresh = function() {
    this.thresh =  127;
    this.alpha =  255;
    this.type = 'Binary';
    this.code = cv.THRESH_BINARY;
    this.otsu = false;
    this.triangle = false;
    this.block = 3;
    this.maxval = 255;
    this.c = 2;
    this.adaptiveM = false;
    this.adaptiveG = false;
    this.adapt = cv.ADAPTIVE_THRESH_MEAN_C;
};
```

An example for the change listener for changing the threshold value is as follows, it uses the variables from the bin object which is of type Thresh.

```
var bin = new Thresh();
  ...
b_threshold.add(bin, 'thresh',
    0, 255, 1).name('Thresh').onChange( function(val) {
    let mat = cv.imread(imgElement);
    ...
   cv.threshold(mat, mat, val, bin.alpha, bin.code);
    cv.imshow('outputCanvas', mat);
    mat.delete();
    ...  });
```

The first two args for the threshold function is the source and destination. Writing mat -> mat in place reduces memory instead of creating a new canvas to write to.

3.2 Milestones

3.2.1 *Milestone 1.*
I started by creating a web page which can upload an image from a file and display it on the screen. I added a Dat.GUI with 1 button which would convert the image to gray-scale.

3.2.2 *Milestone 2.*
Figued out how to implement the change listener for Dat.GUI parameters. and added binary threshold.

3.2.3 *Milestone 3.*
Changed UI and added export button to write to file.

3.2.4 *Milestone 4.*
Added folders and functions for additinal threshold, colors, filters, morphology and contours.

3.2.5 *Milestone 5.*
Increased performance by writing to same Mat. Added contour area and down dample for large images..

3.2.6 *Milestone 6.*
Added pixelation and incorporated XTK.

3.3 Challenges

- Challenge 1: Working with the large opencv.js file I had to load asynchronously and write code in a callback once the file was loaded.
- Challenge 2: Incorporating file io.
- Challenge 3: maintaining multiple canvases and image states.
- Challenge 4: Adding enough parameters to be able to experiment with many different features.
- Challenge 5: Incorporating XTK and WebGL.

4 RESULTS

The following examples show examples of 2d processing and 3d pixel representations of images.

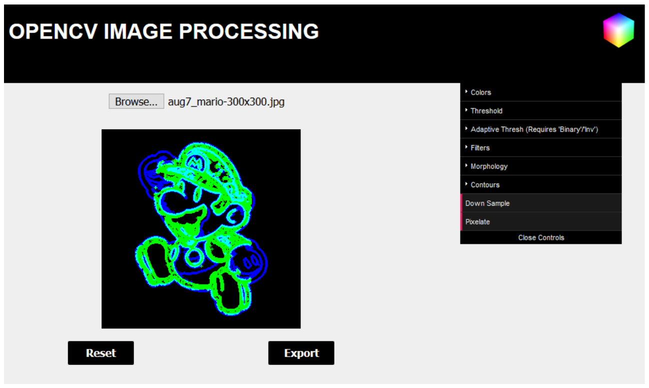

Figure 2: Composition of HSV, binary Threshold, and get structuring element.

5 CONCLUSIONS

The resulting program works well in most situations. the option to export a file is nice since it can be re-imported and use as a future base. The pixelation offers a nice effect and can create interesting visualizations. Performance issues may exist on higher pixel resolutions without a high performance GPU.

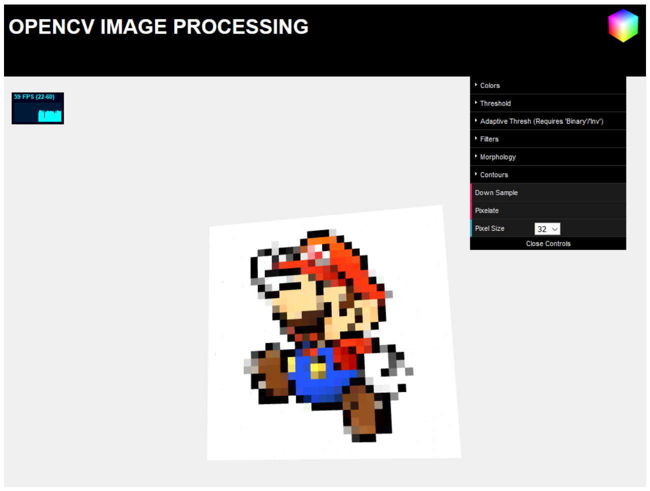

Figure 3: Turn any image into Pixel Art.

REFERENCES

[1] Ricardo Cabello et al. 2010. Three.js. *URL: https://github. com/mrdoob/three.js* (2010).
[2] Daniel Haehn, Nicolas Rannou, Banu Ahtam, P. Ellen Grant, and Rudolph Pienaar. 2012. Neuroimaging in the Browser using the X Toolkit. *Frontiers in Neuroinformatics* (2012).
[3] Open source. 2000. OpenCV. *URL: https://docs.opencv.org/4.1.1/d5/d10/tutorial_js_root.html* (2000).