The Mosaic Project

CSS487 – Cody Stuck, Alex Golovin, Brady Pascoe

# Project Goals

Our goal was to create a program that would create a mosaic image using a library of images that would be laid over an input image. The program should not have any 'bleeding cells', that is, any mosaic tiles that hang over an edge of the image. While the program should create mosaic tiles that are perfectly square, it should be able to handle an input image that does not have square dimensions. The user should be able to specify what size they would like the mosaic tiles to be in order to have control over effective mosaic resolution. An additional goal was to design and implement efficient algorithms so that the program does not waste time or memory (considering there is no limit on how large the input image is, nor how large the image library should be). Finally, it was decided that our program should generate grayscale mosaics instead of color as a proof of concept (color input images will be converted to grayscale).

# Out of scope

Given more time, we would like to implement the following features in the future:

- Output of color mosaics

- Ability to specify output mosaic dimensions

- Ability to load images from the web for use as image libraries

# Design Overview

While building our program, we opted for a modular approach, splitting the program into 3 separate parts:

## Image pre-processing

the image may need to be slightly resized. In addition, the image and the image library must be converted to grayscale, and loaded into the tree according to their average intensities.

## Image tree

To improve space performance and avoid possible sparse image library arrays, the series of images are loaded into a binary search tree. This has the added advantage of finding images with nearest available intensities in a time efficiently.

## Mosaic generation

The building of the mosaic involves allocating a grid of 'tiles', before finding the average intensity of each tile and filling them in with corresponding sub-images retrieved from the image tree.

# Implementation Approach

With three group members, the work was assigned as such:

- Alex G. worked on pre-processing.

- Brady P. worked on the tree of images.

- Cody S. worked on making the mosaic.

All members worked on unit tests for their respective components.

# Project Outcomes

Using a database of standard emojis, we were able to create a mosaic generation program that is able to take in input images of any dimensions and create an output mosaic image with minimal resizing. Our program allows users to specify the input file they would like to use, and how large they would like each cell to be, allowing them to control the effective resolution of their mosaic. By using a binary search tree, we were largely successful in our efficiency goals. We were successful in designing our image loading system so that it could be modified in the future to accept other databases of images. In addition, by using the binary search tree, we should be able to make modifications so that the mosaic program can handle color images by leveraging the nature of the HSV color space (the key for the tree nodes would become hues rather than intensities).

# Learning Outcomes

Much was learned during this project. These are some things that we felt were especially valuable lessons.

- The benefits of modular design

- The benefits of unit testing

- How to integrate OpenCV functions/objects into common data structures (such as a binary search tree)

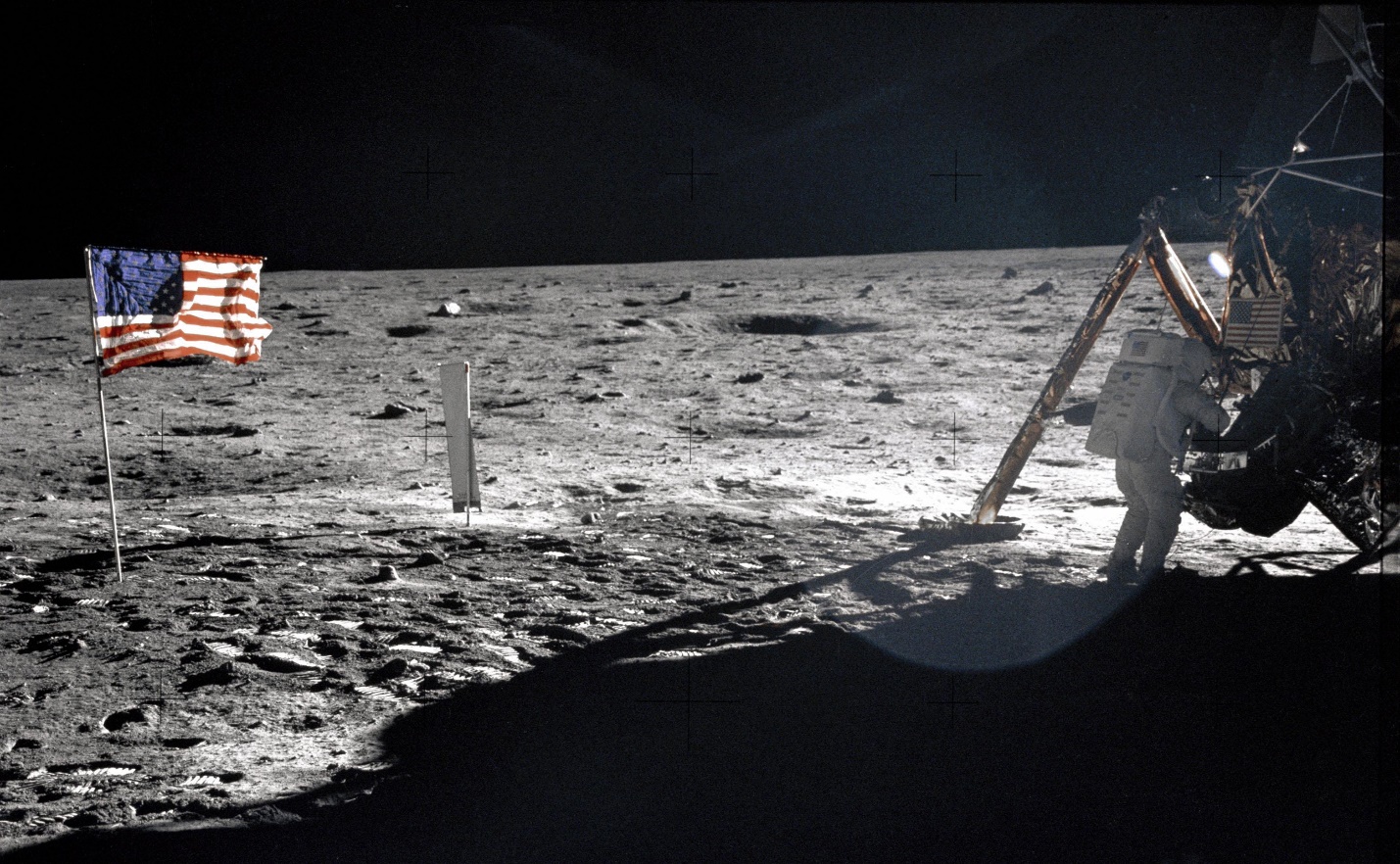
- How to cooperate in a team coding environment

- How to use version control to manage a project between multiple people

- How to collaborate and solve design challenges as a team

# Examples

## Before:



## After: (cell size 20x20 px)