Exploring Color Choice in Art

Emilee Reichenbach
Rensselaer Polytechnic Institute
reiche2@rpi.edu

Casey Conway
Rensselaer Polytechnic Institute
conwac@rpi.edu

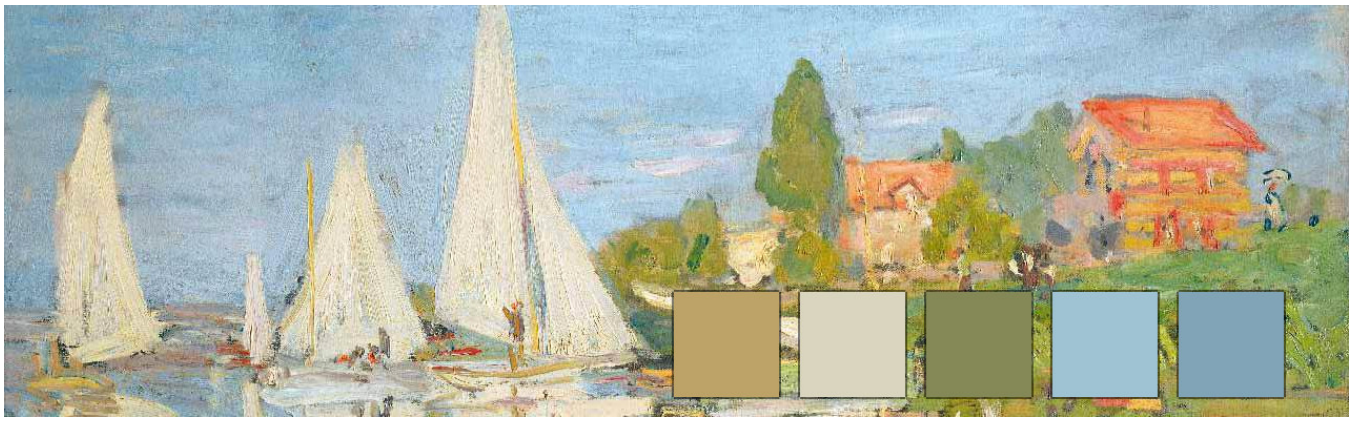

Figure 1. An example image in our database with our automatically computed color palette. *Regatta at Argenteuil - Claude Monet (ID: 19621)*

Abstract

Artists use color to breathe life into their works and choose a palette that best fits the emotion they try to convey. An artist may have a signature palette they reuse throughout their lives or they might conform to the color choices of their particular school or era. Our tool attempts to visualize the historical color choices of 31,000+ paintings throughout Western history and present users with an clean interface to compare and discover the colors used by artists throughout the ages.

1 Introduction

We present a dual interface system for interacting with a large database of paintings. Our initial search page allows users to find paintings based on a variety of metadata with an intuitive and powerful query technique based on modern search engines. This search page allows for the filtering and selection of paintings based on user specified criteria and lets users pick and choose the paintings they wish to compare in a more detailed analysis.

Our second screen provides a comparison view of the images users select for closer study. The primary comparison mechanism of our approach is a radar chart visualizing the distribution of colors in an image. Multiple images can be overlaid on this radar chart to get an idea of how their color choices compare, and can be interacted with to highlight each particular painting's specific details.

In addition to this system, we implemented an automatic color palette generation program to extract color palettes from input images. This k-means based approach produces a color palette of an image to display to the user as inspiration or helpful additional data. These color palettes are displayed along with each image in the search and comparison page, and can be used to get a general sense of the overall colors used in the painting. We also display the hex codes for these palette items to let users pick colors they might want to use in other applications.

This paper will describe the process taken to gather the database of images (Section 3), the steps taken to extract the color information (Sections 4 and 5), and the components of our web tool we developed to make the visualization interactive (Section 6).

1.1 Motivation

The primary motivation for this project stems from browsing Pinterest and finding images coupled with their corresponding color palettes, of which one example can be seen in Figure 2. This pairing is minimal but provides a great deal of insight into what makes a color image work well or not. With a more general interest in art and historical paintings, we developed our research question: are there significant color palette differences or similarities between paintings, whether based on artist, time period, type of painting, or other data? We hypothesize artists will have a similar color palette across their own work, but these similar palettes will slightly change based on the type of painting, time period the work is created in, or art movement the artist associates with at the time. And when comparing two artists, we believe there will be a large palette difference between the two if the

artists' associated art movements, time periods, or schools of art differ.

Figure 2. An example of a motivating color palette found on Pinterest [4].

The target audience for this visualization is anyone generally interested in art or color theory. Our goal is to create an interface that is intuitive and encourages exploration for a wide range of users. By providing users with a powerful toolset to select and compare different paintings, we hope they can develop their own insights about the palettes and color choices of artists throughout history.

2 Related Work

2.1 Color Spaces

To generate the color palettes and Radar Chart data for each painting, we first need to understand how different color spaces can be useful.

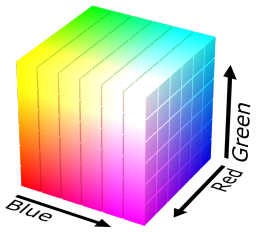

Figure 3. Geometric representation of the RGB color space, where the black point is diagonally opposite of the white point [7].

The first and most common color space we come into contact with is the RGB (red, green, blue) color space. Geometrically, this space can represented by a cube whose axes are the RGB primary colors, with the point furthest from all axes being white, and the point closest to all the axes (the origin) being black. A depiction of this is shown in Figure 3 [9]. This space is common in web development and is the intermediate space we work with when decompressing the original images and preparing our output color codes.

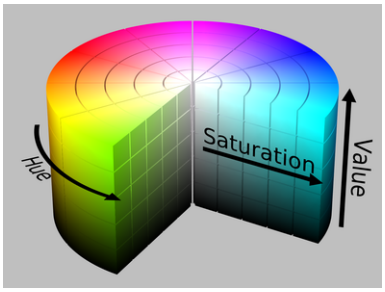

Figure 4. Geometric representation of the HSV color space, where the top base is the maximum saturation for all the hues [8].

The next color space we need to understand is HSV (hue, saturation, value). Geometrically, this space can be represented by a right circular cylinder, whose angular dimension is the hue, radius is the saturation, and axial component (height) is the value, as shown in Figure 4. The top base of the cylinder is a circularized chromaticity diagram, the bottom base of the cylinder is black, and the cylindrical surface contains the maximum saturation for all the hues [9]. The primary benefit of using HSV is the ability to sort colors by hue as it is an effective tool to group various colors together (e.g. all the "blue" colors and all the "red" colors). We use this sorting in our radar chart computation (Section 4). Like RGB, it is still difficult to directly compare two colors in this space, which leads us to our third and final space, CIELAB.

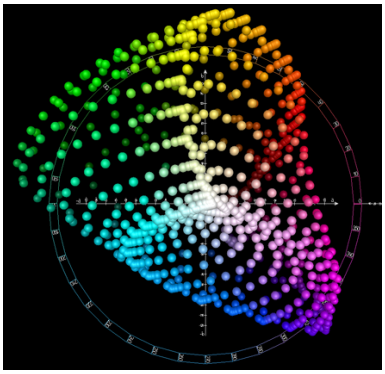

Figure 5. Top view of the CIELAB color space [5].

The CIELAB (or CIE L*a*b*) color space is the most difficult of the three color spaces to understand, but is useful because it defines a Euclidean distance metric upon colors to serve as a mathematical description of how close two colors are

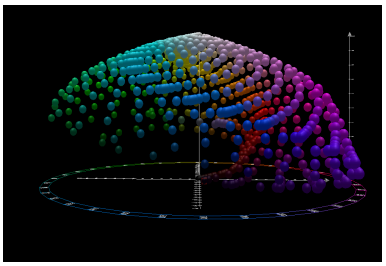

Figure 6. Front view of the CIELAB color space [6].

perceptually. The space is composed of three coordinates, the color's lightness, its position between red and green, and its position between yellow and blue [1]. A top and front view of the CIELAB color space are shown in Figures 5 and 6. We use this space extensively in our K-means palette algorithm (Section 5), as we require the ability to assign each pixel to its closest perceptual cluster.

2.2 Color Palette Generation

There have been many papers written and extensive study of automated color palette generation. One of the more useful resources (which also took a k-means approach, although we developed the details of our technique independently) is from Chang et al. [3] which primarily focuses on recoloring photos by altering their primary palette. They include a detailed description of their technique to produce the initial palette automatically, which includes some useful insights into fixing some of the problems with our technique.

3 Data Acquisition

Our project requires a number of extensive preprocessing steps to generate all the data required by our front-end visualization and search engine components. The computation of color palettes in particular is a moderately expensive operation, especially given the large collection of images we handle. In our particular execution, we followed three main phases of data collection: a scraping phase to generate a database of painting metadata and download links, a downloading phase to acquire local copies of all the images, and a parsing phase to extract useful color information from our entire collection.

3.1 Scraping

Our collection of paintings comes from the Web Gallery of Art (WGA), an online collection of Western art in the public domain from the medieval period to the late nineteenth century. Their database contains more than 50,000 JPG digitized photographs of artwork, complete with information on the artist, technique, and other necessary information pertaining to the particular piece.

Of the 50,000 images, roughly 31,740 of them are paintings (although we noticed a few are mistakenly labeled, with

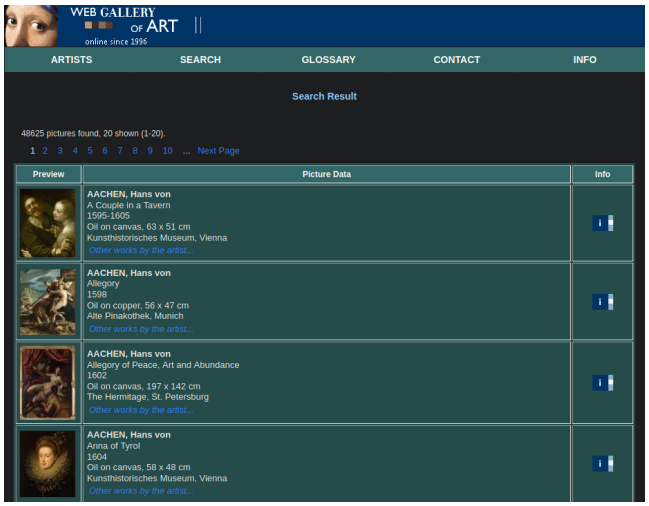

Figure 7. The Web Gallery of Art – our primary data source for acquiring the metadata and image files – and an inspiration for our final interface design.

our local collection containing a number of marble statues and photographs of architectural achievements). The WGA includes a downloadable CSV file containing the entirety of this metadata, but no direct links to the JPG images they host.

Our first step after obtaining this large CSV file was to write a number of SQL queries to filter the data into a manageable form to process later. We removed some of the more superfluous metadata which we decided not to expose in our application and retained only the records marked as paintings. Of these remaining records, we decided to split the remaining database into smaller groups (with a maximum size of about 3,500) based on the first letter of the artist's last name.

This gave us 26 simplified CSV files (although we note that there were no artists with X as the first letter of their last name in this particular collection) containing links directly to an informational page on the WGA along with the metadata we needed for each image. We also added in a unique ID to each record to make it easy to refer to specific paintings.

The final step was to convert the links to the informational pages into direct links to the hosted JPG images. This was accomplished using a simple regular expression thanks to the consistency of their website organization.

3.2 Downloading

With the CSV files we produced, we were able to write a relatively simple Rust script to download this entire collection to local storage. We attempted to be ethical with our use of bandwidth as to not overload their system. This was accomplished by running the entire download process serially with a five second delay between each image. Overall, each image

took about six seconds to complete with the whole process taking roughly 60 hours total.

The download script parsed through the processed CSV files and piped the direct JPG links into WGET. We stored each download image on disk organized by artist last name and image ID. After completion of the download, we uploaded the full set to RPI's Box to exchange the files directly between the two of us to avoid doubling up on the exploitation of their bandwidth (since we both needed local copies).

Our local collection takes up roughly five gigabytes of space, with each image being a compressed JPG file taking up only about 250 kb. These images are not extremely high definition, with the average width being less than 1000 pixels wide, but they are suitable for display in a browser setting and for the extraction of color information.

Figure 8. An example of one of the downloaded paintings in our collection. *Outing in Spring - Bela Ivanyi Grunwald (ID: 14624)*

3.3 Parsing / Extraction

With local copies of our 31,740 images, we were able to proceed to our main processing step: the extraction of color information from each image. Although our program to extract this information was written to be single threaded, we were able to leverage the split of our CSVs into 26 different files to process them in parallel, cutting the total processing time down to about a quarter of the total.

The palette algorithm in particular has an "embarrassingly parallel" piece of computation, which if properly leveraged, could speed up the palette computation to be essentially real time for an image. However, we chose to not implement this image level parallelization and instead focused on the much simpler method of just running multiple sets at the same time. This ended up working very well for our particular needs and required no extra effort writing multi-threaded code.

This extraction program takes in the CSV files as input, uses the unique ID to find our local copies of the images, and performs two separate computations on each image. The first piece computes the necessary data for our radar chart, while the second and more expensive piece extracts the color palette. These processes are described in detail in the following sections.

The program produces a JSON file with the original metadata and the additional color information for use by our front-end site. According to our log files, the computation of these color details required 27 hours and 18 minutes of computation time. With our approach to parallelization, we were able to run multiple copies of this program at a time, shortening the overall color processing time to about 8 hours of heavy CPU use.

When combined with the original querying and downloading stages, acquiring all the data we display in the front-end piece required about 70 hours of computation in total. Writing the code necessary to automate this effort and achieve these results took an additional 20-25 hours. Because of the large amount of effort required to bring these pieces together, we started with a small set (40 images from artists with the last name starting with 'Q') to design the front-end piece, and were able to add the additional data points as we finished processing.

4 Radar Chart Computation

Our front-end comparison page (described in detail in Section 6) displays a summary of the colors used in a painting in radar chart form. This is achieved by computing a histogram of the HSV colors into a few specially selected bins. These values are normalized into percentages of the overall image and output in an array of floating point values corresponding to each color on our chart.

4.1 Histograms

Computing the histogram requires first converting the RGB pixels of the image into their corresponding HSV values. We then loop through all pixels and assign them to a bin based on specific hue breakpoints. We include a special bin for black (gray), which we assign colors to if their saturation or value components are beneath a certain threshold. Experimentally, we found a decent breakpoint to be 20 (out of 255) for both to filter out the grayish tones of an image into their own category. The hue information is very difficult to determine perceptually with such a low saturation and brightness.

In hindsight, we likely should have performed similar checks for the lighter pieces of an image (closer to white), but we were mostly focused on making sure the darker shades common in a large percentage of our collection were grouped together in our chart.

The remaining non gray colors were placed into six other bins: Red, Orange, Yellow, Green, Blue, and Purple. These bins were chosen semi-arbitrarily to reflect a large portion of the HSV hue spectrum without over cluttering the final

radar chart. The actual breakpoints between each of these groups we arbitrarily chose based on our own perception of the colors. The library we used to convert between RGB and HSV colors returns a hue in the range of [0, 180) so the cutoffs we chose were Red (0-8) and (171-180), Orange (9-20), Yellow (21-32), Green (33-81), Blue (82-127), and Purple (128-170).

Using these values we were able to construct a histogram with the total number of pixels in each group.

4.2 Normalization

As constructed, this histogram cannot be compared between images due to the variations in the dimensions of our collection. Thus, a simple normalization step was required to make the data points all on the same scale, regardless of the number of pixels in the image.

This was accomplished simply by dividing the count of each bin by the total number of pixels in the image, resulting in percentage information about how much of an image portrays a certain type of color. These floating point values were stored in our output JSON for use in our front-end.

4.3 Limitations

There are a number of problems with this approach. Our original goal was to show a smoother shape in the radar chart, but due to the way our discretization having only seven categories, we end up with sharp spikes. While these spikes can work well for identifying the colors in a single image, it can be difficult to compare spikes between multiple images.

Our color bins were also chosen relatively arbitrarily. We did not want to over saturate the radar chart with too many categories, but it may have been better to test additional configurations to see if we could produce something more effective. Our palette algorithm constructs a more detailed histogram in the initialization phase, and it may have been worthwhile to attempt to store and display that data instead.

5 Palette K-Means

We used a k-means based approach to automatically extract color palettes from the input images. The technique is based on finding five different colors that reflect the majority of the image. In L*A*B* space, the Euclidean distance between two colors corresponds to the perceptual similarity between them. Colors that are closer together (e.g. Red and Orange) will have a smaller distance estimate than colors that are more perceptually distinct (e.g. Red and Green).

The mathematical property of this color space enables us to cluster our data points (in this case, colors) into five distinct groups. The mean colors of each of the five groups corresponds to our final palette. The goal of k-means clustering is to construct a segmentation such that the groups contain roughly the same number of pixels, which means that all colors in an image will be evenly represented by some component of the final palette.

This technique produces adequate results for the vast majority of our input set. There are some more complicated techniques described in our related works section which fix several of the issues (detailed further in Section 5.4), but for something we designed and implemented on our own it works satisfyingly well.

The code to extract the palette information is written in Rust with the assistance of OpenCV for color space conversion and image manipulation. We uniformly scale down the input image to a maximum width (or height, whichever is larger) of 500 pixels to make processing a bit faster but with virtually the same results. Our non-parallelized implementation takes no more than 6 seconds to run per image, with the cost contingent on the number of k-means iterations required to converge. We limit the maximum number of iterations to 100, but most images require no more than 20.

5.1 Initialization

K-means based algorithms require a set of seed points to initialize the iteration. We originally attempted to randomly select pixels in the image, but received poor and radically inconsistent results based on which values were selected. Our current attempt uses a histogram based approach to try and select colors of different hues to start the algorithm.

Similar to the radar chart computations, we construct a histogram and assign each pixel in the image to a bin. In this version, we have 20 different bins uniformly distributed across hues in HSV space, with an extra bin for gray values with the same parameters as we used previously. Pixels are assigned to bins based on the degree of their hue, with no regard to their saturation or value outside of the gray test. Ignoring the saturation and value parameters may be a mistake and requires more analysis.

Once the histogram is filled with the actual color contents (not just raw counts as before), we take the five most populated bins and use them to compute our starting parameters. We convert all colors in these bins from HSV to L*A*B* space, and then compute a mean color for each bin. These values determine our starting palette.

Picking the most populated bins instead of attempting to find the actual peaks is another mistake, and one of our sources describes in more detail the improvements these changes are able to achieve. More details are contained in section 5.4.

5.2 Iteration

K-means iterations takes a set of input “cluster centers” and returns a more accurate set as output. At each step, we attempt to discover better center locations (palette colors) that more closely align to the goal of uniformly describing the overall image.

This is accomplished by taking every pixel in the image and assigning it to the cluster center it is closest to in L*A*B* space. This segments the image into five different sets of pixels. Each set then computes its mean color in this space, and returns that center as the new output value.

K-means has been studied extensively in the past and has nice properties that work well for this particular use case. Over time, the cluster centers tend to space out and evenly reflect the final image. The iteration shifts the centers around so that every pixel in the image is close to at least one cluster center.

5.3 Termination

As K-means is an iterative algorithm, it needs some sort of termination criteria to prevent infinite loops. As palette colors are output as integer RGB values, it was very easy to determine when the algorithm makes no progress, as the palette after iterating can be compared to the input palette and checked to see if anything changes.

For additional safety, we set a maximum number of iterations to 100 to try and prevent poorly behaved iterations from never converging. No painting in our dataset reached this upper limit as most images converge before 30 iterations. The worst performed image took 73 iterations, but still only 7 seconds to compute single threaded (Fig. 9). More analysis is required to understand why different images may perform worse than others.

Figure 9. The painting with the largest iteration count in our collection – *Francesco d'Este by Rogier van der Weyden (ID: 31160)* – which took 73 iterations to compute a converged palette.

Figure 10. One of the shortest iterations in our collection – *Portrait of a Gentleman by Jacopino del Conte (ID: 6077)* – which took only 3 iterations to compute a converged palette.

5.4 Limitations

There are a number of drawbacks to this approach. The primary issue stems from our palettes ending up being a mean value of different groups of pixels. This results in a muddy, pastel color that is on average close to many different shades of color. This causes us to lose the highly saturated regions of a painting unless they exist in abundance.

Part of this issue may be mitigated by selecting better initial seed colors. Our histogram selects the most populated bins, but this is not exactly what we want as the different peak locations are much more interesting and important to the overall image. It is often the case that one section of the color histogram is extremely tall such that nearby hues are all picked. This leads to the loss of other colors in the image, which may have smaller but just as important peaks. Chang et al. [3] describes a much better technique for computing peaks of a histogram as a means of initializing their seed locations.

There are also some failures to our algorithm. Although we attempted to filter out our database to just paintings, several marble sculptures made it through based on their mislabeling in the original data set. These sculptures (and other pure black and white images) fail to segment the image into five different colors. The breakdown of the technique when there are not five colors in the overall image is expected, but it can be jarring to end users to see a black and white image produce a single gray and four bright blue values for a palette.

Most, if not all, of these drawbacks can be improved with more effort beyond the scope of this project. Improving the palette extraction technique would result in a more useful interface than our current results, but what we have currently is still impressive.

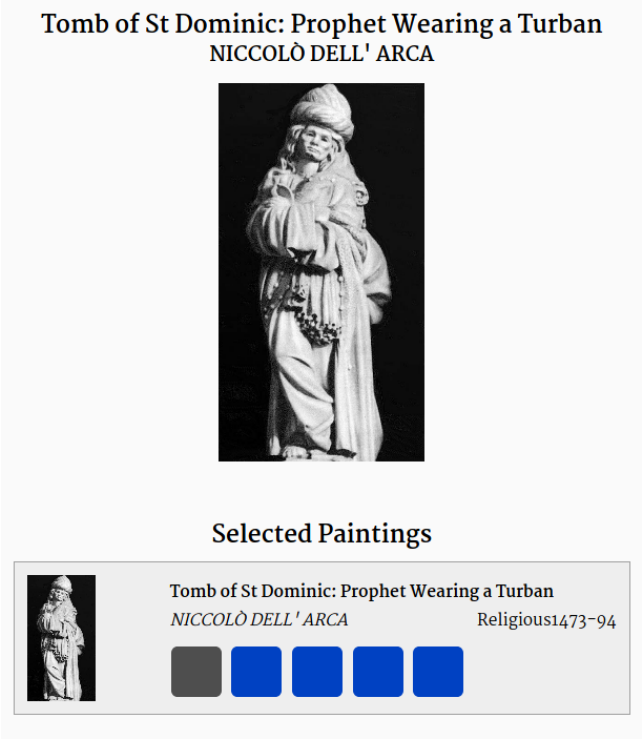

Figure 11. Our k-means algorithm fails to work on pure black and white images.

6 Design Choices

When beginning to turn our idea of color palette comparison into a visualization, we landed on the idea of using a radar chart to present these comparisons. This choice influenced us to lay out our program into two separate pages, one for searching/exploring and one for the actual comparison between paintings.

6.1 Initial Design

We decided from the beginning it would be most effective to allow users to search for paintings separately from the actual comparison, as seen in our two initial storyboards, Figures 12 and 13. From here we created two layouts made from HTML, CSS, and JavaScript, shown in Figures 14 and 16. These screenshots are our intermediate step between our initial design and the final design. After implementing the original design, we made a few design changes based on what we could and could not achieve due to the technologies we decided to use. An example of this is seen in our radar chart design. We were not able to achieve the "grayed out" look in the implementation of our original design, so we decided to do the inverse of this. Although we ended up straying away from our initial design, the final result matched quite closely to the original.

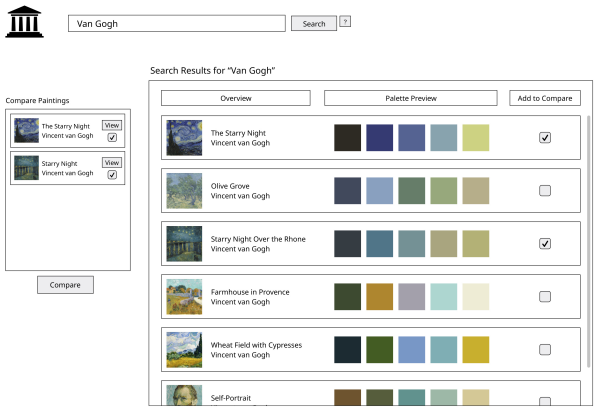

Figure 12. Initial storyboard of the Search page, created in Figma. The final design reflected this closely with some minor changes.

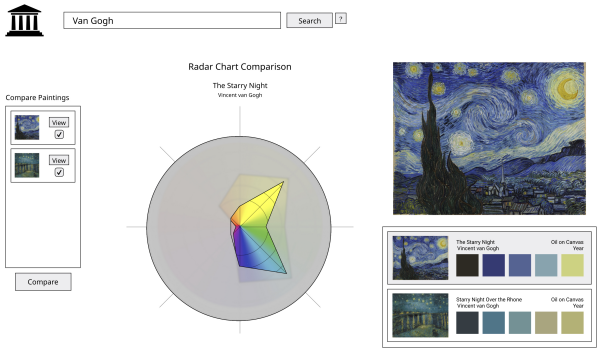

Figure 13. Initial storyboard for the Compare page, created in Figma. The final design reflected this closely with some changes to the radar chart and element placement.

6.2 Search Page

The layout of the final Search page consists of the main title, home button, search bar, help button, paginated search results, and a sidebar containing all the paintings a user selects to compare, as seen in Figure 15. In the search results, the user is shown the total number of results from their search or by default, all of the paintings. We have implemented a pagination system to ensure we do not take too many images at once from our data source's website and to keep the page length to a minimum. For each painting displayed in the search results, there is a thumbnail of the painting, its title, artist, date, technique, and size of the painting in the Overview section. In the Palette Preview section, users can see the generated palette for the painting. And in the

last column, the Compare column, users are able to check a painting to add it to the Compare Paintings sidebar.

In the Compare Paintings sidebar, a user can decide to compare the selected paintings by clicking on the 'Compare' button. This will take users to the Compare page.

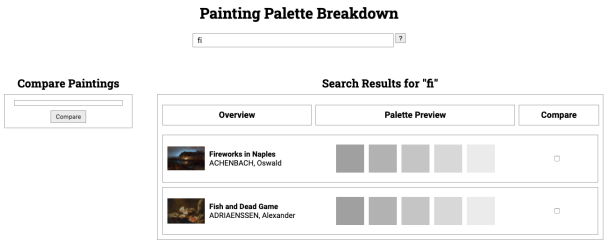

Figure 14. Intermediate design layout of the Search page, between our original storyboard design and our final layout.

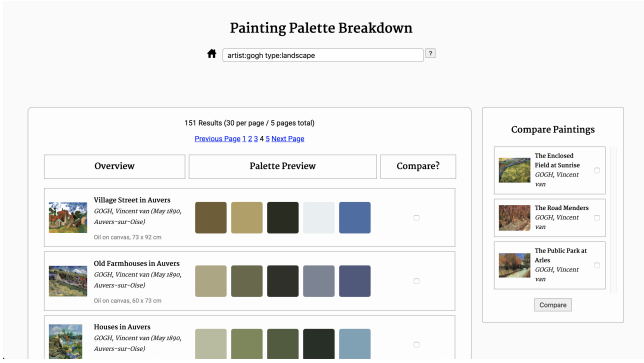

Figure 15. Final layout of Search page.

6.3 Compare Page

On the Compare page, all the paintings that were selected from the Search page will appear in the Selected Paintings sidebar on the right. This is done by using HTML5's 'localStorage' to store the paintings selected on the Search page and access the data on the Compare page. The layout of the Compare page, as seen in Figure 17, consists of the title, home and help buttons, the radar chart, the focus painting and the Selected Paintings sidebar. The radar chart is the page's main focus, while the larger image of the "focus painting", the painting highlighted in light gray in the Selected Paintings sidebar, is the page's secondary focus. We decided to define a "focus painting" to allow users to create a reference painting they wish to compare specific details about to the other paintings. This "focus painting", in theory, could be changed by clicking on a different painting from the sidebar, but it is currently not implemented.

In addition to the "focus painting", the Compare page offers users the ability to interact with the radar chart by hovering over different portions of it. The specifics of that interaction are described in the Visualization section.

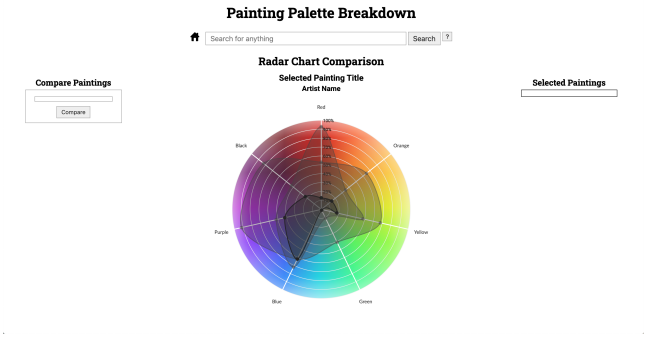

Figure 16. Intermediate design layout of the Compare page, between our original design and our final layout.

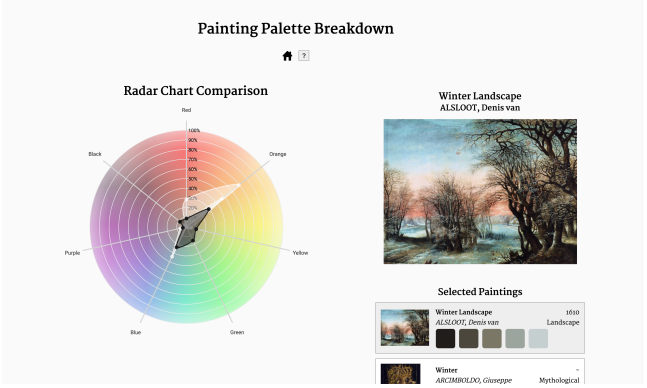

Figure 17. Final layout of the Compare page.

7 Search Engine

To allow our users to effectively search for paintings they are interested in comparing, we had to make sure our search engine provided enough search specificity to users.

7.1 Keywords

We decided to mimic Google's keyword search engine capability by giving users keywords to narrow the results of their search with. These keywords include: 'title:', 'artist:', 'timeline:', 'technique:', 'type:', 'date:', 'id:', and 'iter:'. To use one of these keywords, the user types the keyword and the word or numbers they want to specify their search with immediately after the colon. Currently, we do not support multi-word keyword searches, e.g. 'title:"Vincent van Gogh"',

however this would be a future improvement to make. To achieve the same functionality, users can type 'title:vincent title:van title:gogh' but it is a bit unwieldy. Some keywords are more intuitive than others, such as title, artist, date, and technique, while others require more explanation. The "timeline" keyword is used because not all the paintings have a date, but all paintings have a timeline. This is a 50-year period of time for when the painting was created and can be useful for finding paintings from a general time period or specific art movement. The "type" of a painting refers to whether it is a portrait, landscape, religious painting or some other kind of painting, which our dataset defines the painting to be. The "id" of a painting is a unique id we defined to help identify each painting, and its reason for being used as a keyword is mainly for debugging purposes. The last keyword, "iter" represents the number of iterations our k-means color palette extraction took to generate the painting's palette, another tool for debugging. All these keywords should provide our users with enough control to find what they are looking for, while still having the freedom of not using the keywords.

7.2 AND or OR

In addition to giving users keywords to search with, we decided to use AND logic to help cut down the number of results a user may receive when using multiple words or keywords while searching. In our initial design, we thought using OR logic would be best, so for example, a user could search "Claude Monet" and the results would show both results from the individual searching of "Claude" and of "Monet". After we implemented the logic and tested it using our large dataset, we realized this would likely confuse users and load more results than they expect. This is when we implemented the AND logic to our search bar, which in the end works best with our keywords functionality, so a search can be extremely specific.

8 Visualization

Our program uses a radar chart as its primary visualization alongside a simple palette visualization to provide the user with a large amount of information in a digestible form.

8.1 Radar Chart

We utilized a template radar chart created in D3.js by Nadieh Bremer [2]. The radar chart has 8 axes, each representing a different color category (red, orange, yellow, green, blue, purple, and black/grayscale). The chart's background is an image of a custom color wheel created to align with our axes. This was done to help provide a visual cue to users what each axis means without them having to read the labels. For each painting a user selects to compare color palettes between, there is a blob corresponding to that painting on the chart. By default the "focus painting" has a darker color blob than the other selected paintings. Because it is easy or sometimes possible to determine which blob belongs to which painting, we added tooltips to the radar chart. If a user hovers over any blob, the tooltip will provide the painting's name and artist. If a user hovers over any of the blob's vertices, the tooltip will provide the same information as before, but also includes the exact percentage of the corresponding color the painting contains. An example of this interaction can be seen in Figure 18.

Although the radar chart provides an interesting and interactive way of exploring paintings' color palettes, it has its limitations. Depending on the selected number of paintings or the paintings themselves, the chart can become confusing when many blobs are overlaid on top of one another. As shown in Figure 19, it can be difficult to differentiate blobs from each other if there are a large number of paintings selected. Another issue we have had with the radar chart is the difficulty contrasting our blobs' background colors against the chart's background image. Because we use a wide range of colors in the chart's background, finding colors that contrast well from it becomes difficult and limiting. We have found that using dark and light grays work best overall, because even though black and white would provide the highest contrast, the grays overlap better than pure black or white.

In the future, in addition to the radar chart, we would like to add a histogram either representing only the "focus painting" data or possibly all the selected paintings' data.

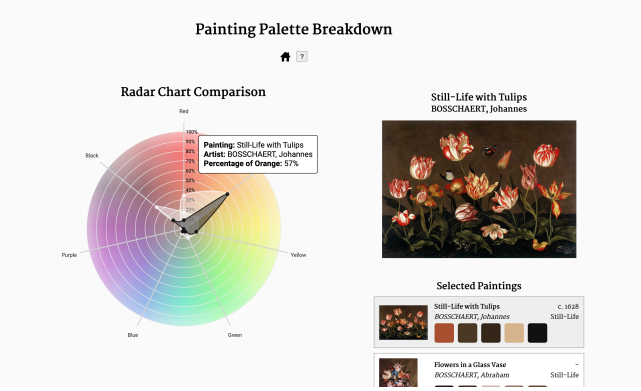

Figure 18. Shown is a screenshot of the Compare page while a user hovers over the "focus painting's" vertex which lies on the orange axis. A tooltip pops up, containing the title of the painting, the artist, and the exact percentage of the color found in the painting.

8.2 Palettes

To visualize the palettes we generated, we decided to use them as a "preview" palette for the radar chart. When users search through all the paintings on the Search page, they are presented with the five squares whose backgrounds are

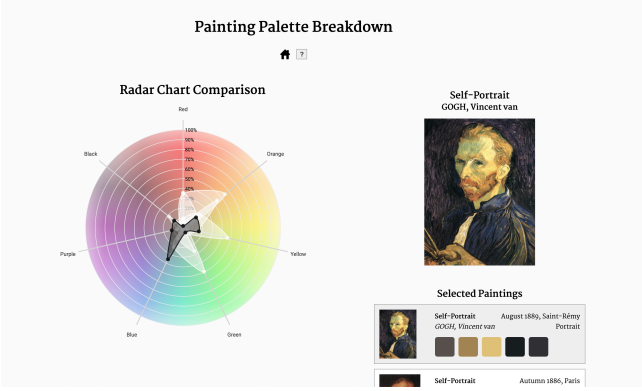

Figure 19. In this screenshot, there are 6 paintings selected and it is clear that it is becoming confusing to differentiate between each of the light-colored blobs.

each a color from the generated painting palette. If a user hovers over the squares, a tooltip with the color's hexadecimal value appears, as shown in Figure 20. Although most of our computation power goes into generating these 5-color palettes, we use them more as a tool to guide users to a final radar chart visualization. However, it may be interesting to further incorporate these palettes into a different kind of visualization.

Figure 20. Shown is an example of the palette tooltip which shows the hexadecimal value of the hovered color, in this case it is the far left, red color being hovered over.

9 Future Work

Along with some minor improvements, there are many directions we could take this project in the future. As far as improvements, we want to complete the color extraction on the remaining images, improve our initial seeds for the k-means palette extraction, get our 'focus painting' feature working correctly, and add a histogram representation of our radar chart to the Compare page to reduce possible confusion caused by the radar chart. A larger improvement that could reduce the likelihood of bugs would be to improve the whole UI by using React.js instead of Vanilla JavaScript.

A potential direction we can take this project is add a "Similar Paintings" feature which could find paintings with a nearby color palette using the CIELAB color space for the computation. We could also use this data to create a force-directed graph, cluster graph, or some other type of visualization to show the nearby palette relationships of paintings. To aid the usefulness of this extension, we would add functionality to allow users to search for paintings that contain a certain color. Because this project is open-ended, there are many other routes we could take to expand upon this project which is exciting.

10 Conclusion

10.1 Feedback

Based on the feedback received from the project presentations, it seems that we have succeeded in creating a useful and interesting tool which focuses on our goals and answers our research question. The main addition to the project users wanted to see was an extension of our search bar to allow users to search by color or by similar palettes. In addition to searching by color or similar palettes, the audience wanted to see another visualization based around how similar painting palettes are to each other. This was also mentioned in the Future Work section above. Overall, we believe we have achieved our goal to create an interface that is intuitive and encourages exploration of the program for a wide range of users.

10.2 Who Did What

The division of labor was split between front-end and back-end work on the project. Both Casey and Emilee developed the initial storyboard for the project. Casey implemented the web scraping, radar chart computation, and palette extraction in addition to any other minor back-end work. Emilee turned the initial designs into functional web pages and implemented the search functionality and visualizations. We divided up the final report in half to each focus on our particular aspects of the project.

10.3 Summary

With this tool, we present an interface for exploring historical paintings and their color palette choices. We have developed an intuitive and powerful search feature to filter our database of 31,000 paintings and narrow down to the images of interest. We present users with an interactive radar chart to visualize the images they choose to analyze in closer detail, and allow users to make judgments about the color choices of artists, eras, and schools of art.

Our final project can be found at https://github.com/emileerei/IntVisFinal. This Github contains our interface and JSON files with the extracted color data. The 5GB collection of paintings is not included, but our JSON files also include direct hotlinks to the JPG files hosted by the Web Gallery of Art, and can be swapped in to allow downloading the images on the fly with some minor changes to our Javascript code.

References

[1] [n.d.]. *Wikipedia*. Wikipedia. https://en.wikipedia.org/wiki/CIELAB_color_space

[2] Nadieh Bremer. 2015. Radar Chart Redesign. http://bl.ocks.org/nbremer/21746a9668ffdf6d8242

[3] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. 2015. Palette-based Photo Recoloring. *ACM Transactions on Graphics* (July 2015).

[4] Jessica Colaluca. 2016. *Design Seeds*. https://www.design-seeds.com/in-nature/succulents/cacti-color-2/

[5] Holger Everding. 2015. *Wikipedia*. Wikipedia. https://en.wikipedia.org/wiki/CIELAB_color_space#/media/File:CIELAB_color_space_top_view.png

[6] Holger Everding. 2015. *Wikipedia Commons*. Wikipedia. https://en.wikipedia.org/wiki/CIELAB_color_space#/media/File:CIELAB_color_space_front_view.png

[7] Michael Horvath. 2008. *Wikipedia Commons*. Wikipedia. https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png

[8] Michael Horvath. 2010. *Wikipedia Commons*. Wikipedia. https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png

[9] George Joblove and Donald P. Greenberg. 1978. Color spaces for computer graphics. In *SIGGRAPH '78*.