Asset Finder: A Search Tool for Finding Relevant Graphical Assets Using Automated Image Labelling

Max Perea Düring
Anton Gildebrand
Markus Linghede
Abstract

**Asset Finder: A Search Tool for Finding Relevant Graphical Assets Using Automated Image Labelling**

*Max Perea Düring*
*Anton Gildebrand*
*Markus Linghede*

The creation of digital 3D-environments appears in a variety of contexts such as movie making, video game development, advertising, architecture, infrastructure planning and education. When creating these environments it is sometimes necessary to search for graphical assets in big digital libraries by trying different search terms. The goal of this project is to provide an alternative way to find graphical assets by creating a tool called Asset Finder that allows the user to search using images instead of words. The Asset Finder uses image labelling provided by the Google Vision API to find relevant search terms. The tool then uses synonyms and related words to increase the amount of search terms using the WordNet database. Finally the results are presented in order of relevance using a score system. The tool is a web application with an interface that is easy to use. The results of this project show an application that is able to achieve good results in some of the test cases.
Sammanfattning

## Contents

1 Introduction 1

2 Background 2
   2.1 Photo-realistic Digital 3D-environments 3
   2.2 Digital Libraries 3
   2.3 Computer vision 4
   2.4 Lexical Semantics 5
   2.5 Stakeholder 5
   2.6 Virtual Reality 6

3 Purpose, aims, and motivation 7
   3.1 Sustainable Development Goals 7
   3.2 Delimitations 8

4 Related work 8
   4.1 Image search 9
   4.2 Pinterest’s Similar Images 9

5 Method, approach and techniques 10
   5.1 Programming language 11
   5.2 Software frameworks 11
      5.2.1 Google Vision API 12
   5.3 Application 13
      5.3.1 Labelling 13
      5.3.2 Search Technique 13
11.3.1 Segmentation and recursion ....................................... 27
11.3.2 Image databases ....................................................... 27
1 Introduction

Recent advances in computing and computer graphics have opened up for the creation of photo-realistic digital 3D-environments in a variety of contexts such as movie making, video game development, advertising, architecture, infrastructure planning, education and graphical design. These environments will be referred to as renderings. This has created a need for rendering tools to aid in the creation of these renderings. These involve big libraries of digital, graphical representations of various kinds of real world objects such as trees and stones or surfaces like snow and hardwood floors, from here on referred to as assets. An asset can for example be a 3D model of a small bush or flower, or it can be a texture such as dirt or concrete.

To achieve an accurate representation of a certain place or environment it is necessary to search these libraries for the most fitting assets. This is often done by trying various key words related to the assets that the user is looking for and a lot of time is spent trying to find the correct words. We complemented this with a tool that allows the user to search using a picture of the place or part of nature that they want to render, and receive search results consisting of relevant assets related to that picture.

Let us say that the user wants to replicate a backyard as a digital 3D render. Instead of typing various words into the search field (fence, rock, old tire) the user decides to use our tool and thus provide it with a recently snapped picture of said yard. The idea is then that a list of relevant assets (3D fence, 3D rock, 3D old tire) is presented to the user in a comprehensible manner. This tool is called Asset Finder.

This kind of image search is made possible by machine learning technology that is available to anyone through various application programming interfaces, or APIs. These are tools that give programmers access to various functions, in this case machine learning based image analysis.

The final result of this project is an application that is able to find relevant assets for certain pictures while struggling with others.

In Figure 1 an example of how the application works is shown. To the left, the user uploads a picture. To the right, relevant assets are presented.
2 Background

In this section we describe a variety of contexts where the creation of digital 3D-environments has become more important and common. We explain what problems they usually face when working with big digital libraries containing assets, such as images, materials, textures and 3D-models. We then present the need for specific rendering tools and why digital libraries are essential for them to work. Further, we will provide more details on why problems appear when working with digital libraries. An introduction to computer vision and its development is introduced to give the reader a picture of why we have chosen Google Vision API for image labelling. We explain lexical semantics and why it is relevant for our project while also mentioning the WordNet database. After that we mention our stakeholder, their problem, and how Asset Finder is a potential solution for them. Finally we mention how renderings can have educational application in the form of virtual realities.
2 Background

2.1 Photo-realistic Digital 3D-environments

In the last two decades there has been a lot of technological developments in the field of computer graphics which has made it possible to create photo-realistic digital 3D-environments in many different contexts, such as animation, entertainment, advertising, infrastructure and architecture. The list could go on, the point is that the need for people to work with this has increased due to the possibilities that photo-realistic digital 3D-environments bring.

To give an impression of the impact they have made we will go through a few examples. First we have the possibility of increasing the demand of products by displaying them using computer graphics. In 2014 IKEA made 75% of the content of their commercial catalogue using computer graphics, only made possible by how close those pictures resembled reality [Par14]. It gives them a big advantage to be able to freely show their products in any environment or setting desired, reducing costs and at the same time getting better results. After that, many other companies have taken their lead. Another context is architecture where “modeling and rendering systems have proven to be invaluable aids in the visualization process, allowing designers to walk through their designs with photorealistic imagery” [DM98]. Finally, in both the gaming and film industry there has been a shift in how visual art is created. The 3D animation market alone is a billion dollar market that is steadily growing [Ana17]. A typical animated movie or game can cost millions of dollars.

The reason why a billion dollar company like IKEA was the first to render the majority of their catalogue or why a movie or game can cost millions of dollars to make is due to the fact that creating visually appealing graphical content often takes a lot of time and resources. One of the reasons for this is that people working with photo-realistic digital 3D-environments usually spend a large amount of time manually searching for assets in big digital libraries [Cha05] [Gan01]. The increase of people working with digital 3D-environments has lead to a growing number of graphical digital libraries containing assets and due to these libraries usually having flawed structures they are not always easy to navigate through text-based searching techniques [Cle98]. The idea behind our tool is to search with an image to find these assets and thus saving time for the growing number of people working with photo-realistic digital 3D-environments.

2.2 Digital Libraries

As mentioned it is possible to create photo-realistic digital 3D-environments in multiple contexts. This has increased the need for rendering tools. These tools are able to generate images from different assets. To be able to use these tools there has been a need
2 Background

of and focus on digitization, meaning the conversion of media and placing it into digital libraries [Cle98], including the assets that people working with digital 3D-environments use in their daily work with any given rendering tool. As mentioned, a common problem for big digital libraries containing assets is that it could be time consuming to find assets using text-based searching techniques. One of the reason for this is that when implementing the structure of a digital library they do not follow a specific standard. Consequently, when working with big libraries it is not always easy to find what the user is looking for. If the user of such a library is looking for written media, such as articles or books, the user can use words that are assumed to be in the title of what he or she is looking for. The user can use a phrase that he or she knows is in the book that he or she is trying to find. It is easy to match words with other words. In contrast, it is much more difficult to use a library of pictures, or as in our case, a library of assets. The user might have a picture in his or her head of what he or she wants but is indecisive of whether it is a stone, a rock or a pebble. Libraries usually have their assets labeled with metadata such as an asset resembling a stone labeled with the word “stone”. If the user then decides that what he or she is picturing is a “rock”, then this “stone” might not appear in the search, even though it might have been perfect.

Since a common metadata standard does not exist the tags for each asset are chosen subjectively, resulting in lacking synonyms and other important information. This makes it more difficult for the users of the library, when searching by text for specific assets [Cle98] [Gan01]. Unfortunately for big digital libraries that are rapidly expanding it is seen as too labour expensive to manually edit all assets into a better metadata structure [Cle98]. We want to solve this problem by using a combination of image labelling and lexical semantics.

2.3 Computer vision

In the late 1960s a goal to mimic the human vision gave birth to the field of computer vision. This field focuses on automatic extraction, analysis, and understanding of images. In recent decades research in computer vision has deepened the relationship between vision and graphics to such a length that automatic detecting, grouping, and extracting of features within an image is possible, which makes image annotation achievable. Image annotation provides descriptive labels for the content of an image. This is usually done by sophisticated machine learning algorithms that are able to extract low-level features, such as colors, shapes and textures. These features are then used to learn what kind of high-level labels, for example forest, house and grass, that the information represent without being explicitly programmed for it [SH19] [Fer19] [Sze10]. The difficulty here is getting accurate labels, which require large amounts of training data and large
2 Background

amounts of computing time [CC17].

Google, who have great computational power, have created the Google Vision API, leveraging this technology. They have state-of-the-art pre-trained models that are able to give image annotations for a big variety of images [SH19]. Given an input image, Google Vision API produces a list of image annotations. Every annotation has a percentage showing the probability of that feature existing within the image. It is also important to note that training a system for image analysis requires many CPU-hours to produce anything useful at all. Since we wanted to use image labelling to solve another problem regarding searching for assets in big digital libraries we chose to use Google Vision API, which offers fairly good accuracy and within the scope of this project it is still possible to achieve fairly accurate results. The labels we get from the Google Vision API are analyzed using lexical semantics with the help of the WordNet database.

2.4 Lexical Semantics

Semantics is a sub field of linguistics which concerns itself with the meaning of language in general whereas lexical semantics considers the meaning of lexical units, which in our case more or less means words. A semantic network is a graph which connects various meanings and their relationships and it can be used to understand similarities between the meaning of words.

An example of a semantic network is the WordNet database. The interesting and useful thing with this database is that it has most English words sorted in hierarchies explaining various relationships between meanings of words. For every word it is possible to determine what words are above, below, at the same level and within the word of interest in the hierarchy. One example is the word forest. Above forest we have the word vegetation, below we have jungle, and at the same level we have woodland and within there is the word tree. This is further elaborated in section 5.3.2. By using WordNet we are thus able to find words that relate to the labels given from the Google Vision API, resulting in more relevant search terms which in turn improves the search results.

2.5 Stakeholder

Our stakeholder, Quixel AB (Quixel), is a company founded in Uppsala. Their main idea is to help graphical designers, today mostly in the movie and gaming industries, by providing them with assets. They saw a need in removing the time consuming process of manually creating assets. They travel around the world to scan different environments and convert them into assets. An asset can for example be a rock, grass, fabric, a brick
2 Background

The assets are sold on a license basis to the film and gaming industry. By using the Megascans library, the creators save time by using ready made assets instead of creating them from scratch. However, their digital library of assets is rapidly growing and the problem of text-based searches being time consuming has emerged for them. Our Asset Finder will work on their digital library to exemplify and show the potential in searching for wanted assets with an image instead of words.

2.6 Virtual Reality

Another important application of computer graphics and 3D-modelling where our tool can be of help is the creation of virtual realities. A virtual reality, or VR, can be accessed by using special VR hardware that the user places directly in front of their eyes to allow them to see into what appears as a different world. This world is generally a 3D-render and thus likely to be made using tools as described throughout this report. Virtual reality can be used as a teaching tool and this was tested by Beijing Bluefocus E-Commerce Co. and Beijing iBokan Wisdom Mobile Internet Technology Training Institutions in a case study from 2016 where the results showed an increase in test results and knowledge retention [Bei16].
3 Purpose, aims, and motivation

In this section a summary of the projects general aims and the reasons behind them are presented. We relate sustainable development goals to our project and explain the vision that we share with our stakeholder. Finally we will mention the delimitations we have decided on and how they affect our work.

The overarching purpose of this project is to explore how we can use existing image-analyzing technology to facilitate working with big libraries of graphical assets. In practice, we are making an application connected to the Quixel library that accepts a picture as input and returns a list of relevant assets. This in turn is an example of how machine learning technology can be used to help graphic artists in their work process.

The aim of this project is to create an application that is connected to the Quixel database and allows search for assets by inputting a picture. The output should be a list of assets ranked by relevance and presented in a clear manner where it is easy to access them. It should also be possible for the user to select smaller portions of the image for which they would like to get suggested assets.

Text-based search techniques can sometimes cause problems as digital libraries of graphical asset increase in size. Using a picture as input can be a more efficient way to describe what kind of assets the user wants to find and everything that can increase the speed of the process of creating 3D environments is helpful.

Since 3D modelling is an important part of creating virtual realities that are helpful in teaching [Bei16] we deem that our Asset Finder has academic value since it can help create virtual realities used for teaching.

Our stakeholder wants to make the world available digitally in the form of these assets and they are interested in exploring the possibilities of converting a picture into a digital 3D environment. To develop a software that suggests relevant assets based on a picture is a step in this direction.

3.1 Sustainable Development Goals

Our project has the potential to make the creative process of creating 3D-environments less time consuming by searching with an image similar to the scene the user would like to create. This could have positive effects on a global scale and we strive to support parts of the two different sustainable development goals below [Uni19a] [Uni19b].
• Goal 8: "Promote inclusive and sustainable economic growth, employment and
decent work for all"

• Goal 9: "Build resilient infrastructure, promote sustainable industrialization and
foster innovation"

By facilitating working with digital libraries containing scanned graphical assets we
make the creative process more efficient. By making it less time consuming to cre-
ate photo-realistic renderings, the possibility for financially independent developers to
make and succeed with their own projects increase. All contexts in society using ren-
dering tools could therefore change into having a more inclusive and sustainable de-
velopment and that in turn would foster innovation. This is due to the fact that if the
creative process using rendering tools takes less time and resources it would be easier
for people with no or little financial strength to create their own photo-realistic digital
3D-environments with all of the benefits and visual appeal that they have.

3.2 Delimitations

The image labelling service that we are using in our project uses machine learning
technology. To create a machine learning model from scratch requires training data and
computing time. With more training data it is easier to create reliable models. Google
have been working on their models for many years and with a lot of resources and thus
they have a level of accuracy in their models that would be nearly impossible to achieve
for a small project such as this.

A way to find assets in a big library through the use of pictures would ideally be done
by training a machine learning model specifically for that purpose. This implies finding
features directly within pictures that correspond directly with features of the assets. For
example a bright picture would correlate to a bright asset. Since this would require huge
datasets and a lot of time and deep knowledge of such technology it is outside the scope
of this project. We circumvent this by using keywords as features to map between the
pictures and the assets and we can thus use the pre-trained Google Vision API to analyze
the pictures.

4 Related work

In this part we will present work that is related in some way to what we want to achieve.
At the heart of our project lies the challenge of searching for things that can not be fully
4 Related work

described by words. This is elaborated in the part about searching for images. Further, we are making a web application that is similar to a lot of existing web applications, such as the one presented in the part about Pinterest’s similar images.

4.1 Image search

An image search searches, as the title suggests, for an image. This is done by using a system that takes in either keywords, an image link or an image file as input and then returns images that are similar to the query. As mentioned it can be problematic when searching for images within a digital library using only phrases or keywords. Reusing the example used earlier you could type ‘rock’ and not find the desired asset with the metadata tagging ‘stone’. In these cases it could be better to look for similarities in queries where the user has an image file or link as input. One technique commonly used for this is Content-based image retrieval (CBIR).

The purpose of CBIR is to use any given image to search for other similar image within a dataset or digital library. The way this is achieved is to first analyze the low-level features of an image such as colors, shapes, and textures. This information is then used to search for similar images within a given visual digital library [CC17] [ZLT17]. This requires that each graphical asset within a digital library has its low-level information stored. This is not the case for many databases, including the Quixel database.

The most obvious example of CBIR is the Google reverse image search. It takes a picture as an input and returns visually similar pictures. The problem that CBIR solves is very similar to the problem that we want to solve, only with solely pictures instead of pictures and assets. The way google reverse image search is implemented is also very different since it uses neural networks trained specifically for this purpose using features such as colors or shapes to link between pictures where we instead use words. Ideally we would have wanted to use similar techniques as CBIR but it is outside the scope of this project. Instead we want to accomplish similar results by connecting keywords and metadata in ways that allows the user to do a similar reverse image search, but for assets.

4.2 Pinterest’s Similar Images

Pinterest [Pin19] is a web service and a social network built around images with focus on inspiring users with a large image catalogue of ideas. Their algorithms continuously present new images to the user, which are thought to be interesting to the user. Pinterest also have the *visually similar images* feature, which allows the user to select a portion of an image and from that selected portion show images which Pinterest think are similar.
This is close to what we are doing, allowing the user to find visually similar assets to selected portions of our input image.

Pinterest’s goal with this particular feature is very similar to what we want to achieve, with the difference that we are finding assets rather than already rendered images.

We can read more about Pinterest’s work with finding visually similar images in their paper *Visual Search at Pinterest* [JLK+]. The authors demonstrate how Pinterest utilizes convolutional neural networks together with image captions to receive accurate suggestions of similar images. By doing this, they managed to increase user engagement while keeping the development and computational costs down.

## 5 Method, approach and techniques

This section presents the methods we used to achieve the goals of our project and why we chose them. The project can be seen as to consist of three different parts. Constructing the actual application is the first part, getting relevant search terms through labelling the input picture is the second part and getting relevant results from the Quixel database by using a good search technique is the last part. The user flow of our application is:
• The user selects a locally stored picture, which the client sends to the server.
• When the server is done processing the picture, the client receives a list of suggested assets as a response from the server.
• The suggested assets are presented with a preview and a name.
• The user can refine the asset identification by selecting smaller regions of the image.

5.1 Programming language

The primary programming language we use for the client and the server is TypeScript. TypeScript is a superset of JavaScript (a just-in-time compiled programming language), meaning that TypeScript is compiled to JavaScript [Mic19]. Using TypeScript helps us during the development with type hints and also helps detecting possible errors in the application through the use of types. JavaScript comes in different versions complying to different language specifications called ECMAScript. We chose to target ECMAScript 6, which has a wide browser support while still giving access to new language features.

One of the reasons for choosing TypeScript was that we wanted a language that we could use for the client as well as for the server. This allows us to be able to focus on doing software development rather than having to learn one more language. Everything that works for JavaScript also works with TypeScript. JavaScript is one of the most popular languages used in 2019 according to the Stack Overflow Developer Survey 2019 [Sta19], and therefore has a large base of frameworks to choose from.

5.2 Software frameworks

Angular [Ang19] was chosen as the framework used by the client. We chose to use a framework because it speeds up development compared to not using a framework at all. One thing that Angular handles is data binding, thus making the presentation of data easier because we do not have to update the user interface manually. Making the presentation of the user interface easier is the main reason for using Angular. We also chose to use Angular because we had some prior knowledge of it. An alternative to Angular is React [Rea19], which is a popular JavaScript library for making UI rendering easier. Angular is a fully feathered framework, while React is a library solving the smaller task of UI rendering. The two projects have in common that they make UI rendering easier for the developer. Angular however simplifies more tasks than UI rendering. These include making HTTP calls, routing and testing.
For the server we chose to use NodeJS, which is a JavaScript runtime that can be used without the use of a browser, for example for server applications. By running JavaScript on the server, as well as for the client, we made it possible to share some code between the two different parts of the project. This gave us more time to focus on developing the algorithms rather than doing the actual implementation. An alternative to using NodeJS would be to use a different programming language for the server, such as Python.

5.2.1 Google Vision API

The Google Vision API is a software developed by Google and is part of the Google Cloud Platform, which is a suite of cloud computing services available for anyone [Goo19]. It takes a picture as input, and produces an output with detailed analysis of the picture. The skills that the Google Vision API features includes are labelling, dominant colors, landmarks and objects. A label describes what the Google Vision API thinks the picture resembles. Each label comes with a score, representing how sure the Google Vision API is that the label is in fact correct for the given picture. Dominant colors is the set of colors that take up large amount of the picture. If the picture contains any landmarks, such as the Eiffel tower, Big Ben and Statue of Liberty, it will return the landmarks geographical location, position in the picture, and name of the landmark. The Google Vision API can also distinguish if there are any objects in the picture and if so it returns an array of objects, their position in the picture, and what type of object it is, such as car, cat and bird. The most relevant functionality for our project is the labels functionality. By receiving descriptive labels of the content within an image we are able to create our search tool for digital libraries with graphical assets.

Some of the advantages of using The Google Vision API instead of our own machine learning (ML) solution is that it comes pre-trained on huge amounts of data and we do not have to do an implementation of a ML solution, saving us time. The downside is that the Google Vision API is not optimized for what we are trying to do, since it is a general purpose image analysis software. For example, the Google Vision API is not trained to recognize the textures we are trying to identify. It can be really good at detecting rocks, but not necessarily what type of rock, which could be relevant for the user.

We chose the Google Vision API because it is easy to use, free for small projects, and by using a pre-trained solution we do not have to train our own neural network. Some alternatives to Google Vision API are Microsoft Azure Computer Vision API and IBM Watson Visual Recognition. These services work similar to Google’s service and using them instead of Google Vision API would yield fairly similar results.
5 Method, approach and techniques

5.3 Application

In this section the inner workings of the application are explained. First we describe labelling followed by a section explaining what search technique is used.

5.3.1 Labelling

We use The Google Vision API for picture labelling. A label is a word or a short sentence describing the picture. These labels can be used to search the asset library. However these labels retrieved from the Google Vision API might not be very good to search the asset library, since the assets might have different meta data than what would match the labels found through the Google Vision API. Therefore, we do filtering and further analysis of the labels. First, we filter out the labels that we think are qualitative for use in a search query. We do so by filtering out labels that do not consist of single words only. We then look for synonyms for the labels, to get alternative search terms. We do this by using the publicly available WordNet [FT19] database, which is a large lexical database of English words. It describes words, relations between words, and makes it easy to find the basic form of a word as well as to find synonyms to the word.

5.3.2 Search Technique

The underlying idea in our project is to create a tool that resembles Content-Based Image Retrieval, which we discuss further in Section 4.1, but that makes use of keywords and metadata instead of directly linking the contents of the input picture and the assets, since that would require a lot of resources. The Google Vision API provides us with a set of up to around 20 labels. These are words that are related to the input picture. These labels each have an accompanying percentage representing the probability of the label being correct.

The asset finder searches the Quixel database for every label. In each of these searches, each asset found gets its score increased (starting at 0) by the percentage that was related with the label. The tool also makes use of the WordNet database to cover more search terms. The original set of labels provided by the Google Vision API is expanded by related words with another percentage implying distance from the original word. A synonym to a label will have the same percentage as the original label. A related word to a low percentage label will have lower score than a related word to a high percentage label. This approach lets us cover more of the Quixel database while also providing more relevant results. After all searches are done, the assets are shown in descending order ranked by their score.
WordNet has an underlying structure where we can find related words through lexical relations called hypernym, meronym, synonym, and hyponym. Hypernym means the super-ordinate relation, that is words being higher up in the lexical hierarchy. Meronym is the part-whole relation between synsets, meaning that synsets could be part of the whole of another synset. Synonym is other words that have the same meaning. Last we have hyponym which is the sub-ordinate relation meaning that it is a word lower in the lexical hierarchy.

To clarify all of the above we can have a look at Figure 4. Since forest is vegetation, vegetation is one of its hypernyms. There are different types of forest, such as bosk, grove, and jungle. These words are hyponyms to the word forest. The words that share the same meaning as forest are called synonyms. Meronym are words which are part of a forest, for example underbrush or tree. By looking at these relations that the meaning of words have we are able to find related words to the labels we get from the Google Vision API. By searching with these related words we increase the probability of searching with the correct keywords and thus making it easier to find the most suitable assets. To make this even more accurate we evaluate words depending on how far they are from the label given by the Google Vision API. For each step \((N)\) away from the given label in the hierarchy we multiply the score, given by Vision, connected with the label with \(\frac{1}{(N+1)}\). An example would be as in Figure 5 where at the fourth step we see the hierarchies, here the word ‘landscape’ would have the total value of 0.33 and ‘vegetation’ would have a value of 0.495. The reason for this is that the label ‘forest’ with the score of 0.99 is two steps away in the hierarchy from ‘landscape’ and one step away from ‘vegetation’. This results in: landscape = 0.99 * \(\frac{1}{3}\) = 0.33 and vegetation = 0.99 * \(\frac{1}{2}\) = 0.495

![Figure 4 Lexical relations and hierarchy given by synset of “forest”](image)

### 5.3.3 Overview

The process of matching an image with assets is as follows (illustrated in Figure 5):
1. The server receives a picture from the client, and sends it to the Google Vision API.

2. The resulting \textit{labels} from the Google Vision API are analyzed, arranged in a hierarchy to each other, and for each label we look for synonyms.

3. We traverse the labels hierarchy from bottom up, and search the assets database for that label. The assets results for each label are saved for further computing.

4. After computing a score for each asset, the result is flattened and returned to the client formatted as JSON [JSO19].
6 System Structure

Our application uses the conventional client server model [The19], where the client sends requests to the server and the server does all the processing. The server then returns a well formatted response to the client, which the client then presents in a way that is intuitive for the end user. The basic system structure is illustrated in Figure 6. The client only communicates with the server. The server communicates with the Google Vision API and Quixel’s asset library, as well as with the client.

![System Structure Diagram](image)

**Figure 6** The client communicates with the server. The server communicates with the Google Vision API and Quixel’s asset library

6.1 Client

The client is what the end users use. It is built as a web application on top of the Angular framework [Ang19], which is a JavaScript framework for simplifying web application development. We have wrapped the client as a desktop application using Electron [Ele19], which is a project that enables developers to ship web applications as if they were native desktop applications. The client communicates with the server over HTTP [Moz19] by making requests over the network.
6.2 Server

The server takes a picture as input. This image is analyzed using the Google Vision API. The labels acquired are then used to search the Quixel asset database using Algolia, which is the search engine that Quixel use for searching their internal database. Finally, a list of assets is returned.

7 Requirements and evaluation methods

In order to evaluate the results of this project and assess how well we have managed to reach our goals, that is, to create a tool that facilitates working with big libraries of assets, we have formulated a set of requirements with corresponding evaluation methods.

7.1 Interface requirements

For our tool to be useful it is important to have an interface with which the user interacts that makes it easy to accomplish the intended task of finding relevant assets. The user is supposed to interact with our application by first providing an image and then by reading the results. Jakob Nielsen provides 10 general principles of interaction design [Nie19] from which we have chosen to focus on Recognition rather than recall, Aesthetic and minimal design and Error prevention. We have chosen these three principles since they are the most relevant principles for a one-page application such as our Asset Finder. The other heuristics lack importance when there is no need to maneuver between different pages.

- Aesthetic and minimal design
  Our application should have no unnecessary elements. The design needs to be simple and clear with no distracting text or graphics.

- Recognition rather than recall
  Our application should be intuitive. There should be no need to learn how to use the application or to consult a manual.

- Error prevention
  Our application should be error-free. The use of our application should not be interrupted by errors.
7 Requirements and evaluation methods

The interface should contain no unnecessary information and have a box where it is clearly signaled that the user can upload an image. There should be no more than one view with which the user interacts so that there is no need to go back and forth between views. The results are to be presented next to the search box so that a new search can be done without having to return to a previous view. Finally there should be no frequent errors when using the application.

The interface of the final application will be reviewed with regard to these requirements and a discussion will assess how well the interface works in accordance with them.

7.2 Relevance of results

The relevance of the results is an important part of our tool to assess since it largely decides the usefulness of our tool. It is also the hardest part to assess since relevance is hard to measure. We decided to use a set of six images to test our application and through discussion assess the relevance of the results through the following criteria, which all have to be met for the result as a whole to be relevant:

- Number of items found.
  This is the most basic criteria since the other criteria can not be assessed if no items are found. If no items are found, the tool has no use, but the finding of items does not imply success since the items found might not be relevant.

- Relevance of items found
  This is the second most important criteria since our goal is to find relevant items. If items are found and at least one of them is relevant we can assess that we have succeeded with creating a tool that can find relevant items, though it might not be very useful if for every relevant item found a big amount of irrelevant items are found. An item is relevant if it can be used to render an environment similar to the input picture.

- Amount of relevant items found
  This is the third measure where the amount of relevant items found is compared to the amount of irrelevant items. The larger the amount of relevant items compared to irrelevant items the more relevant is the result as a whole.

- Coverage of input picture
  This is the final criteria where we assess to what extent the most prominent elements of the input picture are represented among the search results.
The pictures were chosen to cover a variety of landscapes in order to expose as many weaknesses and strengths in our tool as possible. Since the results are ordered by relevance we decided that only the first 100 results be considered.

8 Evaluation results

In this section the results from our evaluation are presented. The six images and a part of their respective results in the application are shown and accompanied by a discussion. This discussion aims to assess the relevance of the results for each picture with regard to the requirements presented in the previous section. In the second part the interface is presented and evaluated according to our previously stated criteria.

8.1 Relevance of results

Each of the six pictures is shown together with a part of the acquired search result. The overall relevance of the search for each picture is assessed through discussion.

8.1.1 First picture

The first picture shows a dirt road in a deciduous forest, dried leaves, and twigs. The results acquired consists almost entirely of relevant items such as twigs, branches and trees. More than 100 items were found and the relevance of the first 100 items is high with the exception of a group of similar items called 'Green Painted Wood’. These items can be seen in Figure 7. The coverage is good since a lot of the items seen in the picture had similar items appear in the results with the exception of leaves and dirt.
8 Evaluation results

Figure 7 The first picture as input to the left, suggested assets to the right. Where we have scrolled down to show a variety of assets

8.1.2 Second picture

The second picture is a close shot of the ground in a forest. Grass, stones, branches and various plants are present. The results are satisfactory, once again consisting of more than 100 results that are mostly relevant such as grass, dirt and a stone. Some irrelevant assets related to sand appeared. No asset related to the branch appeared.

Figure 8 The second picture as input to the left, suggested assets to the right. Where we have scrolled down to show a variety of assets
8 Evaluation results

8.1.3 Third picture

The third picture shows a mountain, a road and forest. The results acquired consists mostly of things related to the forest and contained a lot of mushrooms. No assets related to the road, to the snow or to the mountain were present in the results. The item most resembling a mountain was a smaller rock. Although some of the assets found could be used to render a similar environment, most of the search results are irrelevant.

![Figure 9](image.png)

Figure 9 The third picture as input to the left, suggested assets to the right

8.1.4 Fourth picture

The fourth picture also shows a mountain, but in a desert-like environment. The results consist mostly of different kinds of rocks. Some of the rocks are relevant but most seem to belong in other environments.
8. Evaluation results

8.1.5 Fifth picture

The fifth picture shows a small Caribbean island with some huts and a lot of palm trees. The results consist entirely of seaweed which could possibly be relevant for a render but it is not what is seen in the picture. Most of the items seen in the picture are not represented in the results.

Figure 11 The fifth picture as input to the left, suggested assets to the right
8.1.6 Sixth picture

The final picture shows a house among rocks and some vegetation. The results consists of mostly rocks which are clearly relevant, but they lack assets related to the vegetation and the house.

![Figure 12](image)

**Figure 12** The sixth picture as input to the left, suggested assets to the right

8.2 Interface

The interface can be seen in the previous section and in Figure 13 and is deemed successful with regard to the requirements specified. For the recognition rather than recall requirement we made several users test our application. All of the users that tested the application managed to use it in the intended way by dragging a picture into the input box. They also managed to switch between categories to filter the result and also to click the reset button when they wanted to search by another picture. The applications interface worked just as intended. The interface is minimal and shows clearly how to provide it with a picture thus completing the principle Aesthetic and minimal design. Further, no errors have appeared throughout extensive use of the application. Since there are only two states and few ways to interact with these and none of these create errors we deem the application to be sufficiently free from errors and thus complete our third requirement, Error prevention.
9 Results and discussion

In this section the results of the project are presented as well as discussed. The functionalities of the final version of the Asset Finder are presented, the process evaluated and our accomplishments analyzed.

We have seen that the tool is capable of providing relevant assets when given a picture. The results are varied and the relevance of the results is highly dependent on what kind of picture is provided. The tool seem to work best when given closer and more detailed images. The results are acquired quickly and the interface completes our requirements.

The images that give best result all have in common that they have a lot of visible objects. This makes sense since the Google Vision API is able to identify things such as a rock or a twig and send those labels to the Quixel database, resulting in relevant results. It is also important to note that the Quixel database does not cover all things found in nature and the lack of relevant results can be a consequence of searching for things not present in the database.

Another possible reason for the lack of results is that the Google Vision API’s output can have labels that contain more than one word, which we do not take into consideration, resulting in loss of important information regarding the image content.

In the third picture there are several different regions such as a road, a forest, a mountain and a glacier, but when looking at the results the assets shown can be related mostly to the forest region. This could be showing that our Asset Finder have a tendency to focus
on assets in connection with one region dominant in the picture.

In the fifth and sixth picture we see that one kind of asset completely dominate the results. We could possibly see improvements by limiting results of the same kind.

The island picture is dominated by water, which yields assets directly related to the ocean, in this case seaweed. This is probably due to the Google Vision API giving ocean-related labels, which makes sense since a big part of the picture consists of exactly that. It causes problems, though, since the assets found are related to the ocean in a way that is not relevant to the picture.

The tool works best when used with pictures of nature that have obvious and clear elements that allow the Google Vision API to assign labels with high certainty. Also, it seems that the results are better if the image contains roughly similar elements. If the image contains different sorts of environments the Asset Finder tends to increase relevance for assets only connected to one of these environments.

10 Conclusions

We have developed a tool in the form of a web application that allows the user to search for relevant graphical assets in the Quixel library of 3D-graphics using an image as input. The tool uses image labelling services made available by Google through their Vision API to extract labels from the input image. These are cleaned from unusable labels and expanded with synonyms and similar words through the WordNet database to achieve a set of key words. The key words are used to search the Quixel database for assets and these assets are presented in order of relevance where the relevance is calculated using the score that the labels acquired through the Google Vision API carry with them. The application has been designed to be easily used even by first-time users.

The tool was designed to be of use to our stakeholder and to exemplify how machine learning technology and image labelling can be used to aid working with big digital libraries that are hard to navigate. Our evaluation results have showed that the Asset Finder can succeed in finding relevant assets when fed with a picture. However, there are limitations to what kind of images actually work with the tool and the relevance of the items found is not always satisfactory.
11 Future work

In this project report we have presented a tool that allows finding of relevant graphical assets in a digital library. This tool is specifically designed to work with the Quixel database and it is made to accomplish the single task of searching their library by using an image as input. There are a lot of ways this tool can be further improved and expanded and some of those will be examined in this chapter.

11.1 Generalization

Since the tool we have made is designed to work specifically with the Quixel library it is at this point useless when working with another digital library. To expand the application to include an API or some other way to connect it to other databases would allow more people to make use of the Asset Finder and allow it to be used more generally. These databases would not have to be libraries of 3D-assets as the functionality of searching by a picture can be useful in other contexts as well.

11.2 Assertion of usefulness

We have showed in this report that it is possible to find relevant assets using our software. We have not showed, however, if it is useful to replace or complement traditional search methods with our tool in a 3D-modelling workflow. Investigating the potential improvements in efficiency that our tool could provide through interviews or questionnaires would be a valuable addition to this project.

11.3 Purpose specific machine learning model

As noted in section 3.2 we did not have enough time and resources to train a purpose specific machine learning model for this project. As mentioned in section 4.1 there are ways to search with an image by analyzing low-level features, such as colors, textures and shapes. Training a machine learning model to correctly connect between these low-level image features and similar features of the assets in the Quixel library could improve the results of our application. It would take a long time and possibly a lot of resource to develop such a system due to the difficulty of training a system to analyze specifics in a picture such that it can recognize enough elements to get relevant assets. If completed though, it has more potential than our system.
11.3.1 Segmentation and recursion

Another step to further improve the purpose specific machine learning model would be to implement algorithms for segmentation of the image and then recursively do image analysis to get more detailed low-level features to search with in the digital library. For this we have found that one of the most useful segmentation algorithms called Conditional Random Fields (CRF) which if implemented well could with high accuracy classify each pixel or specific regions in an image [Bus18] [KK11] [AZJ18] [CPK18]. For this to work the CRF learning algorithms have to have predefined classes, such as road, forest, mountain, glacier and air. The classes mentioned are only examples, they are chosen when creating the system and it is important that they explain potential image content.

As an example, if Figure 12 would be analyzed with a CRF system containing the classes above, it is probable that it would be divided in the following way: the road is segmented into one region, the yellow areas on both sides of the road is the forest region, above the forest would be the mountain and/or glacier region, and at the top would be the air region. Then all regions of interest, such as road, forest, mountain and glacier, would be analyzed once again in addition of analyzing the whole image and by doing this, more detailed low-level features are acquired. Using this extra information when searching would result in more accurate and relevant results compared to analyzing the input image once.

When trying to implement this into a machine learning model, if for example the classes tree and forest both exist at the same time, it could be difficult to not confuse these two when learning their characteristics [Bus18]. It could be hard to tell when the segmented region should be forest or tree, which a forest contains. In Figure 7 the desired segmentation for the green area on the top of the picture is forest. The problem is if these two classes exist perhaps it would divide it into several regions for both tree and forest classes, which is unwanted. To solve this issue, hierarchies for segmentation could be implemented.

11.3.2 Image databases

When training a machine learning model it is interesting to look at databases such as ADE20K [MIT19]. They provide an image database for interior and urban environments that are pre-segmented and split into hierarchies. One example of a hierarchy is the top class called house which in turn has second classes like steps, doors and windows, where the class windows in turn could contain third classes such as pane and shutter. This is an example of a three layer hierarchy. Because of ADE20K’s hierar-
chies it is possible to get more information from a picture containing any of their top classes. It is then possible to get segmentations of the picture at various levels of the hierarchy. All their hierarchies have 1-4 layers of classes [ZZP+16][ZZP+17]. For our Asset Finder a database like ImageNet containing 14 million images following the hierarchies and synset networks in WordNet would be the most suitable database because it contains natural environments and objects. However, ImageNet does not divide each image into hierarchies as ADE20K does, it only has one label for each image and all labels are stored in the WordNet structure [DDS+09]. A combination of ADE20K and ImageNet would be optimal to be able to in great detail segment and see assets within an input image of natural environments. For example in Figure 7 it would be desirable to first segment the top part as forest, then within that segment the trees. Further segmentation could be the tree trunks, bark, branches and foliage. By doing this it is more likely that an image search, when analyzing those specific parts, provides better results when searching for graphical assets compared to analyzing the whole picture.

References


References


