Master's Thesis

Design and Implementation of a Configurable Generic Search Engine Indexing using Scalable Crawlers

Alhajras Algdairy

Examiner: Prof. Dr. Hannah Bast
Advisers: M. Sc. Natalie Prange

Albert-Ludwigs-University Freiburg

Faculty of Engineering

Department of Computer Science

Chair of Algorithms and Data Structures

April 25th, 2021

Writing Period

01. 12. 2020 – 25. 04. 2021

Examiner

Prof. Dr. Hannah Bast

Second Examiner

Prof. Dr. Thomas Brox

Advisers

M. Sc. Natalie Prange

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Web search indexing is an essential system that powers modern search engines. It automates the process of collection and organization of data from web pages to create an updated index of the web that can be optimally searched. Web search indexing consists of two essential components, a web crawler, in which search engine bots systematically traverse the web to find new or updated content based on rules declared beforehand, followed by the second component which is the indexing of the collected data. The process of web search indexing comes with its own challenges, including performance, managing dynamic content, and answering the question of what is the most relevant content. As the web continues to evolve and grow, the task of web search indexing will remain a key focus of search engine technology and research.The aim of this thesis is to design and implement a generic configurable web search indexing that can be used as a basic tool on different websites and can be further expanded and improved, and scaled. The approach included a simple UI design that allows users to configure and create crawlers and index the generated data.

Acknowledgments

First and foremost, I would like to thank:

- My parents for supporting me during the master's program.
- My wife for her love and support.
- Prof. Dr. Hannah Bast for accepting my topic and for her guidance and supervision.
- M. Sc. Natalie Prange for her thoughtful ideas and suggestions.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Since the beginning of the Digital Revolution, known as the Third Industrial Revolution, in the latter half of the 20th century, the importance of data has increased as it became the new currency shaping the dynamics of our interconnected world. From social media platforms and e-commerce transactions to information sharing and entertainment consumption, online activities generate enormous amounts of data. The online data is sometimes referred to as the 'new oil' or the 'new currency', as it impacts almost the same economies and societies as oil. Businesses and organizations understand the power of data as they provide insight into consumer behaviour, refines business strategies, and enhance decision-making processes. Furthermore, the rise of artificial intelligence has further amplified the value of Internet data. Natural language processing (NLP) is becoming a new hot topic as all the giant firms race to create their model; however, data is the fuel to power those models. The more data is collected, the better the model can become. Consequently, collecting, analyzing, and leveraging internet data has become a cornerstone of competitiveness, innovation, and progress in the digital age.

The Internet data can be harvested by using automated software programs called Web crawlers, also known as web spiders or web bots. Their main goal is to discover, retrieve, and index information from websites.

Internet data can be harvested by using automated software programs called Web crawlers, also known as web spiders or web bots. Their main goal is to discover, retrieve, and index information from websites. The applications and use cases of internet crawlers are diverse and valuable, to name a few:

- Search Engines: Crawlers are essential components to build any search engine, such as Google, Bing, and Yahoo. Crawlers are run on supercomputers to crawl all the content on the internet index web pages and gather information about content, keywords, and links. This data is then used to rank and display search results, ensuring users can quickly find relevant information.

- Market Research: Businesses use web crawlers to collect data about their competitors, market trends, and consumer opinion. This information helps in making informed business decisions.

- Fraud Detection: Cybersecurity companies use crawlers to catch fraudulent activities by monitoring online transactions, identifying unusual patterns, and tracking potential threats.

- Content Monitoring: E-commerce platforms utilize crawlers to extract product prices from various websites. This enables them to offer consumers real-time price comparisons and assist in finding the best deals. Moreover, social media platforms use crawlers to monitor their content to prevent unwanted posts and images.

1.2 Problem Statement

The World Wide Web (WWW) contains enormous data that escalates with each passing day. The total data created and replicated is expected to grow to more than 180 zettabytes by 2025 according to Statista. This upward trajectory is expected to continue due to the growing affordability of smartphones and the broadening reach of internet accessibility. Moreover, due to the COVID-19 pandemic, more companies started offering remote work, more local shops transformed into online stores, and more services switched to cloud-based. This social evolution over recent years has embedded the Internet as an integral cornerstone of our daily life.

The expansion of the Internet gives rise to an immense overflow of data, resulting in a noise that complicates the task of locating relevant information for both end users and organizational queues. To surmount this hurdle, the concept of Information Retrieval (IR) was coined by Calvin Mooers in 1951. IR involves the art of accessing and recovering data from an extensive pool of unstructured information. A particularly pragmatic manifestation of IR involves the extraction of data from the Internet, thus advocating the implementation of a universal algorithm for procuring and categorizing requisite information. In this pursuit, crawlers or spiders emerge as automated entities designed to adhere to predefined directives, allowing the automated fetching and extracting of data from the Internet.

One form of IR is a web search engine. A web search engine is a system engineered to index the Internet. Users can search for articles, documents and pages by entering keywords. The search will provide a list of the most related result that matches the search query. Using the crawlers explained earlier; the engine can index the collected information and optimize the search process using different algorithms and techniques.

Although search engines such as Google, Bing and DuckDuckGo display remarkable proficiency in their web crawling and indexing capabilities, specific businesses, like those in E-commerce, have a distinct interest in demonstrating the most competitive pricing for a given product, a key insight into their competitive landscape. However, more than a straightforward Google search is needed, as the search query index and rank the documents on the Internet based on Google's vendor parameters. These parameters include preeminent brand visibility, user geolocation, SEO optimization proficiency, and hidden variables, excluding the lowest price criterion from the page ranking equation. A second issue arises from the format of search results, with each search engine providing a distinctive result format. Companies may want to exclude some portion of the Internet from the index and rank. Also, they should prioritize some pages more than others.

The previous requirements are tiny use case, among others, that limits companies from using a simple search on Google. Search engines need to be tweaked and configured to match the domain of interest as E-commerce in the previous example and to match a specific use case like the price comparison mentioned.

The main concern is that businesses are often interested in only a portion of the Internet that intersects with their domain and expertise. Furthermore, the criteria for indexing and page ranking depend heavily on their use case and is vital to their business to take control of it and configure it as fit. Hiring domain expertise is inevitable for any business. However, the data scientists often have to go throw some basics steps to get their crawler up and running; those steps cost money and time; it would be helpful to have an infrastructure that allows the data scientist to have starting script that can be extended easily and needs little to no programming knowledge.

Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025 Published by Petroc Taylor , Sep 8, 2022 https://www.statista.com/statistics/871513/worldwide-data-created/: :text=The

1.3 Contribution

In this thesis, we aim to answer the following questions:

- What are the challenges and bottlenecks to creating a scalable, configurable generic search engine?
- Can we implement a basic tool that can be easily scaled to crawl different websites independently from their DOM structure?
- Can we create a proper User Interface UI that allows users to crawl and index targeted websites from the internet?
- Can we integrate the indexing and crawling processes in the same tool?
- Can we find meaningful evaluation metrics for the implemented search engine?

1.4 Chapter Overview

2 Related Work

The concept of web crawling dates back to the early 1990s when the World Wide Web was still in its infancy.

WebCrawler, created by Brian Pinkerton in 1994, is considered the first true web crawler-powered search engine. While some may claim that title for Wandex is due to its potential, it was never designed to be used in this way. Wandex lacked some critical features to make it a general-purpose search engine.

One of the major innovations of WebCrawler was its full-text searchability. This ability made it popular and highly functional. It continues to operate as a search engine, although not as popular as Google, Yahoo, Bing, Yandex, or Baidu.

Modern web crawlers face many challenges and complexities, such as dynamic content, user interaction, authentication, robots.txt files, and ethical issues. Some examples of modern web crawlers are Googlebot, Bingbot, and Internet Archive

2.1 Web crawlers list

Web crawlers have evolved during the last few decades, with different designs and implementations to crawl and index the internet. Below is an enumeration of some of the architectural designs utilized in the development of all-encompassing web crawlers [3]:

- RBSE [Eic94] Considered as one of the first Web crawlers to be published. Made of two components: the first component, “spider”, uses a queue in a database. The second component, “mite”, is a modified browser that downloads the pages from the Web.

- WebCrawler [Pin94] The initial publicly accessible full-text index of a specific portion of the World Wide Web was established. The approach involved leveraging lib-WWW for page downloads and employing an additional tool to parse and arrange URLs, ensuring a breadth-first approach to navigating the web graph.

- World Wide Web Worm [McB94] was a crawler designed to construct a basic index comprising document titles and corresponding URLs. This index could be queried by utilising the grep command in the UNIX operating system.

- Google Crawler [BP98] Google has been the market’s dominant search engine for the last few decades. In March 2023, Google’s global market share was 85.53The crawler was integrated with the indexing, and since this thesis has some similarity with the Google search engine design, we will explain this in-depth in the ext subsection [Google architecture]

- Ubicrawler [BCSV04] Is a Java-based distributed crawler with no central process and several identical “agents”. The crawler is implemented to provide high scalability and be tolerant of failures.

2.2 High Level Google Architecture

The Google search engine’s design gives a good overview of the essential components to create a scalable search engine. Hence, it is a great starting point for any search engine research; we will explain it in this section. Most code written in the Google search engine was implemented in C or C++ for efficiency and because it can run on either Solaris or Linux [2]. Google uses distributed crawlers to download internet web pages. The URLserver keeps a list of the available found URLs that need to be crawled by the crawlers. URLserver acts as a load balancer that sends the URLs to the following free crawler. Afterwards, the crawlers download the documents needed from the page, associate a unique ID for this page called docID, and then the page's content are stored in Soreservers. Storeservers then compress the pages and save them on a repository. The indexer component then uncompresses the pages and parses them. Each document is then converted into a set of words called hits. The hits represent the word and its position in the document. Afterwards, the indexer distributes those hits into barrels. Moreover, the indexer collects links found in the crawled page and stores them in the anchor's file. The anchors' file contains the links and their relationship with each other [2].

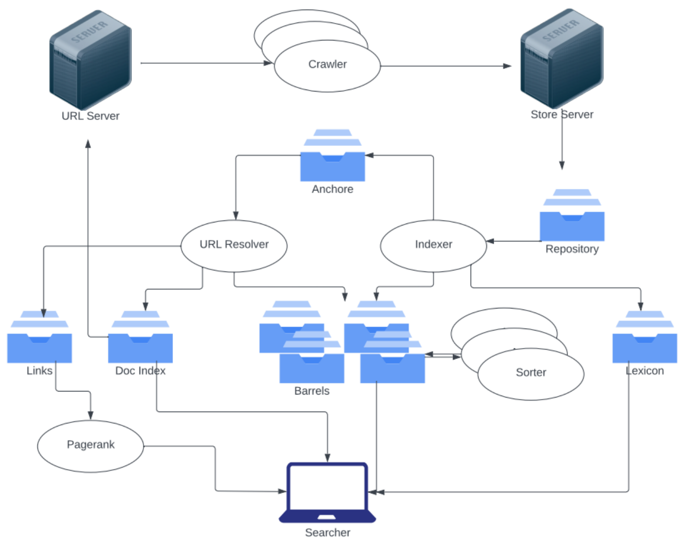


Figure 1: High level view of Google web crawlers archeticture.

The URLresolver reads the links from the anchors' file and converts the relative URLs into absolute URLs. The URLs are then assigned to their docID. The links database saves pairs of docIDs that will be used to compute PageRanks for all the documents.

Initially organized by docID, the barrels are then rearranged by the sorter based on wordID. This process generates an inverted index. Moreover, the sorter generates a list of wordIDs and corresponding offsets within the inverted index.

2.3 User Friendly Crawlers

Since this thesis also focuses on providing a friendly user interface to configure the search engine, it is advisable to look at the existing tools in the market. Although there are plenty of tools to parse the content of the internet without programming knowledge, two of the most well-known software are Octoparse and ParseHub. Since ParseHub has a free tier to test it and it supports Linux, I decided to go with it to evaluate my crawler and collect some ideas on how they approach the different challenges with configuring how to use the crawler for various websites.

ParseHub is a user-friendly option well suited for users who want a straightforward User Interface tool to parse information from a specific website on the internet. It operates as a desktop application, catering to multiple platforms, including Linux, Windows and Mac OS X. Comparable to Octoparse, ParseHub adeptly handles intricate web scraping tasks outlined earlier. Despite its aspiration for simplicity in web scraping, users with particular technical understanding may need to delve into its more advanced features and grasp their intricacies [7].

2.4 Performance Evaluation

Evaluating a web crawler can be challenging as it is usually made of different components, like the web crawler and the indexer. However, in this section, one crucial aspect of the evaluation is evaluating the overall design and architecture. As introduced in the conference of [6], the following issues made the distributed parallel crawlers challenging:

- Overlap: Overlap happens when numerous simultaneous processes engage in downloading pages, potentially resulting in multiple instances of the same page being fetched by different processes. This situation arises because one process might not detect if another process has already visited the page. This results in redundant downloaded pages that should conserve network bandwidth and enhance the overall efficiency of the crawler. The question then arises: how can we effectively orchestrate these processes to mitigate overlap?

- Quality: A crawler frequently prioritizes downloading "important" pages to optimize the overall quality of the gathered content. However, within a parallel crawling environment, individual processes might need a comprehensive overview of the results of the other processes. Consequently, each process could base its crawling choices solely on its limited web perspective, potentially leading to suboptimal decisions.

- Communication bandwidth: As already explained that the crawlers try to increase their quality. They must communicate with each other. However, for a large number of processes, the communication overhead can be problematic. One should reduce the communication to only the critical information that can increase the quality with minimizing the communication overhead.

- Scalability: Those targets can contain millions of pages, even focused crawlers that only crawl one site, like Reddit, Amazon or StackOverflow. As mentioned, in certain cases, a single-process crawler cannot achieve the required download rate for huge sites. Hence scalability is necessary to overcome this challenge.

- Coverage: denoted as c/u, is represented by the ratio of pages crawled (c) to the overall pages (u) explored by the entire crawler. In an error-free scenario, an optimal coverage would be 1. No duplicated pages will be stored and visited, increasing efficiency [5].

3 Background

This section tackles the fundamental principles and groundwork of the theory encompassing concepts, terminology, and methodologies related to search engines as applied within this thesis.

3.1 The nature of the web

The web we know today, Web 2.0, is known as "the participative social web" and is massive, and its rate of change is enormous due to its highly dynamic content. Due to this big sample space, finding the relevant pages or documents from the web isn't easy. To overcome this issue, there are two main approaches to sampling: Vertical sampling: Focus only on the pages that are restricted by the domain name. This can be done in different levels. For example, one can restrict the crawling process based on the country, such as .de, .ly, uk. When vertical sampling is done at the second level, it limits the crawling to domains (e.g. stanford.edu) [3]. Horizontal sampling: In this approach, it is important to know the most visited pages and domains to keep crawling from them, giving them more priority than others. This can be done by collecting data logs from the ISP or utilising a web crawler to estimate the page's popularity [3].

3.2 History

The World Wide Web is an unlimited space to share provide and share information. Those information can have different format and cover different doamins. The use case of the web is only limtied by the developers imagination. This is benifital as the Web keept evolving rapidaly form Web 1.0 to Web 2.0 to Web 3.0. Web 1.0 used static pages to serve information, those information were moslty news, blogs and personal langing pages. Some refre to the Web 1.0 as "the read-only web". Although Web 1.0 was massive however most content were created was by deverlopers or at least users who knew basics of the HTML and CSS, moreover by that time content were only static they did not depen on fancy JavaScript libraries and frameworks like Angular and React, this made it limited to some use cases only. Fast forward, pages become more dynamic after using sessions, databases and clint rendering schemas. Those changes made the Web focused not only reading and gathering information by gave the power to more audiounce who did not know any programming or coding to participate and interact with the Web via browsers. Social media, e-commerce and trading stocks platforms was one of the reasons made the internet buble inflate, Use cases where unlimited as useres could create and deploy their own websites by using simple tools as Content Manament System CMS. This made Web 2.0 known as "the participative social web".

To optimize the allocation of crawler resources, estimating the page's freshness must be considered. This prevents outdated pages from remaining unrefreshed for prolonged periods or where lesser-significant pages are needlessly recrawled despite unchanged content.

It can be understood intuitively that the likelihood of a copy of page p being up-to-date at time t, denoted as up(t), declines over time when the page is not revisited.

$$u_p(t) \propto e^{-\lambda_p t} \tag{3.1}$$

The parameter p signifies the rate of modifications occurring on page p, and its estimation can be deduced from past observations, mainly when the web server indicates the page's last modification date during visits. Cho and Garcia-Molina derived this estimation technique for p [8].

$$\lambda_p \approx \frac{(X_p - 1) - \frac{X_p}{N_p \log(1 - X_p/N_p)}}{S_p T} \tag{3.2}$$

- Np number of visits to p.
- Sp time since the first visit to p.
- Xp number of times the server has informed that the page has changed.
- Tp total time with no modification, according to the server, summed over all the visits.

Note that some pages do not include the last-modified time stamp, and in this case, one can estimate this manually by comparing the downloaded copies at two different times and using the following equation. Where Xp now will be the number of times a modification is detected.

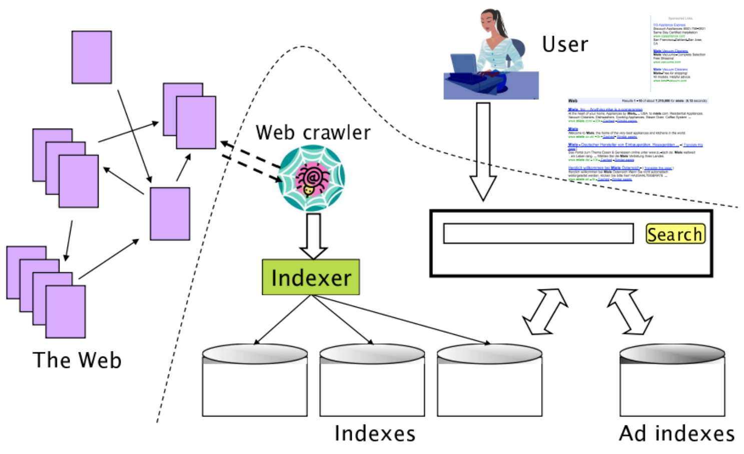
$$\lambda_p \approx \frac{-N_p \log(1 - X_p/N_p)}{S_p} \tag{3.3}$$

3.3 Web Search Engine

Web Search Engine is software that collects information from the web and indexes them efficiently to optimize the searching process by the user. Users enter their queries to ask for information. The engine performs queries, looks up the pre-built organized index, and returns relevant results. The returned result is presented by

Search Engine Results Pages as known as SERPs. The result is then ranked based on predefined criteria.

Web search engines use web crawlers or spiders to collect and harvest the internet jumping from one page to another. Each page can contain several links. The crawler's task is to find the links, visit them, and harvest them. Followed by crawlers, indexing is the next process where information is organized and optimized for search.



3.3.1 Requierments and Features

Search engines, regardless of their imp[lmentation and design there, are some features and requirements that make a good one; following is a list of the most fundamentals features:

- Web Crawling and Indexing: Each search engine needs two main big components, Crawler and Indexer. The Crawler is the component responsible for collecting pages and downloading them from the web. An indexer is used to create an index to facilitate efficient searching.

- Ranking and Relevancy: The algorithm determines the order in which search results are presented to users based on relevance.

- User Interface: The user interface where users enter their search queries and view results.
- Scalability and Performance: Distributed Architecture: A distributed system helps handle the vast amount of data and traffic. This needs a Load balancer to distribute the cralwing tasks betrween the nodes and threads.
- Data Storage and Management: A robust database system is necessary for storing indexed data and metadata.

3.4 Cralwer

Web crawler or spider is a software which gathers pages information from the web, to prived the necessary data to the Indexer to build a search engine. The essential role of crawlers is to effectively and reliably collect as much information from the web. This thesis invests more time on this component than the Indexer as it serves as the bottleneck to the Search engine performance.

3.4.1 Cralwer Specifications

Crawlers can have a wide vireity of features and specifications, however some are necessary to include and others are vital to have a reliable useable one. More information can be found in the book [9]

- Robustness: Web crawlers can be fragile and easy to break; this is due to the nature of the dynamic contents on the web and the internet connection. Web crawlers must identify those edge cases and obstacles and tackle them.

- Politeness: The implementation of the crawler can be unintentionally Mellitus and dangerous if not designed correctly. A Denial of service DoS and a Distributed Denial of service DDoS attacks can occur due to a bad crawler implementation. Hence crawlers must respect websites policies and avoid breaking up web services and loading the servers.

- Distributed: For optimal efficiency, the crawler should possess the capacity to operate in a distributed manner across numerous machines.

- Scalable: The crawler's design should enable the expansion of the crawl rate by seamlessly integrating additional machines and bandwidth.

- Performance and efficiency: The crawling system should adeptly utilize various system resources, including processor capacity, storage capabilities, and network bandwidth.

- Quality: The crawler should be biased towards fetching "useful" pages first.

- Freshness: Acquiring recent versions of previously accessed pages. This is particularly relevant for search engine crawlers, ensuring the search index remains updated.

- Extensible: Crawler design should possess extensibility across numerous dimensions, facilitating adaptation to novel data formats, emerging fetch protocols, and similar challenges. This necessitates a modular architecture that accommodates expansion.

3.4.2 Crawler architecture

The simple crawler architecture is made of the following fundamental modules, as shown in the following Figure. Fetch module that communicates with the internet and collects the pages passed by the URL Frontier module using HTTP requests from the URLs. URL frontier module contains a list of the URLs that need to be fetched by the Fetch module. Parsing module that takes the page content found by the fetch module and parses the page content to find the following links to be passed to the URL frontier and also to parse any value needed from the page, like text and images. Duplication filter that is used to exclude seen URLs. The DNS resolution module is responsible for identifying the web server from which to retrieve the page indicated by a given URL [9].

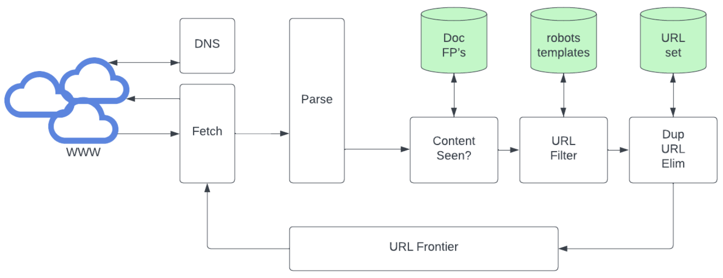


Figure 2: The basic crawler architecture.

The first step is to add a seed URL to the URL frontier. This URL works as a starting point for crawling. The crawler then fetches the page corresponding to the seed URL and stores it to be parsed by the parser. Subsequently, the page undergoes parsing to extract both its textual content and embedded links. The content will be used by the indexer component in the search engine. Moreover, each identified link by the parser module is subjected to a set of evaluations to determine its eligibility for addition to the URL frontier.

After finding the future links and content by the parser, filtering both found links and content is needed. The first step is to check if the page content has already been seen; this can be done by checking the page content fingerprint. The most straightforward method is to use a checksum (stored in the Doc FP's). The next filter is to exclude the parsed new URLs. The URL filter will run some tests to exclude unwanted URLs. This can be some URLs out of the country target, like .de, or some restricted URLs that should not be visited by the crawler. Excluded

URLs list can be added manually to the filter. However, there are more rules written by the domain admins that should be followed. Those rules can be found under a standard text file named Robots Exclusion Protocol (robots.txt).

"robots.txt" acts as the selected filename for implementing the Robots Exclusion Protocol, which is a widely adopted standard employed by websites to signal to web crawlers and other web robots the specific sections of the website that are permissible for them to access [10]. The "robots.txt" can be fetched at the starting point of crawling and can be cached through the whole crawling process, as it can be assumed it will not change during the crawling process. This assumption is still better than making an HTTP request to get the robots.txt file for each URL that needs to be fetched, as this will duplicate the number of requests and reduce the crawler efficiency and also load the server with unwanted requests. Including the robots.txt in the crawling, process should be mandatory as this will serve the point about politeness mentioned in the Crawler specifications section.

3.5 Indexing

2.4 Web crawling issues page 27 book Effective Web Crawling by Carlos Castillo

4 Approach

The approach usually starts with the problem definition and continues with what you have done. Try to give an intuition first and describe everything with words and then be more formal like ‘Let g be ...’.

4.1 Problem Definition

Start with a very short motivation why this is important. Then, as stated above, describe the problem with words before getting formal.

4.2 First Part of the Approach

4.3 N-th Part of the Approach

5 Datasets

6 Experimental Evaluation

7 Summary of Results

8 Conclusions and Future Work

Bibliography

[KimathiKimathi2020] Kimathi2020Kimathi, G. 2020June. What Is Ray Tracing Technology and How It Works in GPUs. What is ray tracing technology and how it works in gpus. https://www.dignited.com/62084/how-ray-tracing-works