FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

ODPR

Offene Datenbank für Physikalisch Basiertes Rendering

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Andreas Wiesinger
Matrikelnummer 01429087

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr. techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Christian Freude

Wien, 3. März 2020

Andreas Wiesinger	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

ODPR

Open Database for Physically-based Rendering

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Andreas Wiesinger
Registration Number 01429087

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr. techn. Michael Wimmer
Assistance: Dipl.-Ing. Christian Freude

Vienna, 3rd March, 2020

Andreas Wiesinger	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Andreas Wiesinger

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. März 2020

Andreas Wiesinger

Kurzfassung

Die Ausbreitung von Licht und seine Wechselwirkung mit Materie kann näherungsweise mit mathematischen Modellen simuliert werden, den sogenannten bidirektionalen Reflexionsverteilungsfunktionen (englisch Bidirectional Reflectance Distribution Functions (BRDFs)). Jedoch stellt das Modellieren phyikalisch korrekter BRDFs eine Herausforderung dar, ebenso wie die Verifikation ihrer Korrektheit. Verschiedene Herangehensweisen wurden bereits entwickelt um photorealistische Render-Algorithmen auf ihre Korrektheit zu untersuchen. Das Problem ist jedoch, dass diese Methoden und Test-Szenen nicht ausreichend von der Community genutzt werden. Eine Voraussetzung für die Durchführung von Tests ist das Vorhandensein von Testdaten und -methoden. Eine weitere ist die Bereitschaft der Community, diese auch zu nutzen und Tests dann auch tatsächlich durchzuführen. Diese Arbeit befasst sich mit der Förderung von letzterem. Für diesen Zweck wurde eine Web-Applikation namens “Open Database for Physically-based Rendering (ODPR)” entworfen. ODPR stellt einen zentralen Sammel- und Zugangspunkt für verschiedene Test-Szenen für unterschiedliche Test-Methoden zur Verfügung. Ein Prototyp für ODPR wurde entwickelt. Die Applikation baut auf Community-basierten Design-Mustern, ähnlich zu StackExchange-Seiten, in denen die Nutzer*Innen selbst die Wartung und das Hochladen von Test-Szenen übernehmen. Die Idee ist, dass die Community selbst zum Wachstum und zur Wartung von ODPR beiträgt indem die Test-Szenen frei heruntergeladen werden können und registrierte Nutzer*Innen von zusätzlichen Berechtigungen profitieren können.

Abstract

The propagation of light and its interaction with matter can be simulated using mathematical models, most commonly Bidirectional Reflectance Distribution Functions (BRDFs). However, the creation of physically accurate BRDFs and their verification can be challenging. In order to be able to test and verify physically-based rendering algorithms, various methods have been researched. However, they are rarely used by the community. One key to the verification of rendering algorithms is to provide test-methods and test-data. Another key is to motivate the community to actually use them and run more tests. This thesis focuses on the latter. For this purpose, the author designed a web-application called "Open Database for Physically-based Rendering (ODPR)", where test-scenes of different types and from different studies will be merged into one publicly available place. A prototype for ODPR was implemented. The web-application uses community-driven design-patterns similar to StackExchange-sites, and allows scientists to register and upload test-scenes. The idea is, that ODPR will be built up and maintained with the help of the community, by providing free downloads of test-scenes and additional privileges to registered users.

Contents

CHAPTER 1

Introduction

There are many photo-realistic rendering algorithms [1]. However, proving the correctness of physically-based rendering is still a challenging and open problem in Computer Graphics (CG). Different aspects hinder the CG community to perform the necessary analyses [2]: The amount of new verification techniques and test-scenes grows. The scenes and techniques are often published in different places. Physically based verification methods are cumbersome to carry out. Especially ground-truth data: Real physical objects are photographed with precise measurement tools. Each photograph is taken from a different angle. Many potential interferences need to be considered and taken into account. Hence, the scene needs to be very minimalistic. To completely capture a material one needs to measure as many lighting and viewing configurations as possible. Each measurement is time consuming and captures only a fraction of the complete parameter space. Therefore, it is important to use measuring pipelines which keep the number of necessary measurements at a minimum. O. Clausen et al. [3] therefore focused in their study on the color and the reflection model: They investigated diffuse color patches and how they reflect light under different incidence and reflection angles (Figure 2.1). Such ground-truth datasets, consisting of a few thousands of recorded samples, can then be compared to their rendered counterparts. The render-predictions are based on a precise model of the scene from the ground-truth dataset. Each light source and each object needs to have the same position, rotation, surface and shape. These alignments can be tedious and prone to errors but make it possible to calculate the deviation of the rendered images from the physical photographs.

Because of these drawbacks, not only ground-truth datasets [3, 4, 5] have been made, but alternative test-scenes and verification metrics have been researched as well [1, 6, 7]: Render predictions can not only be compared to ground-truth data, but to other render predictions as well. This alternative approach, in comparison to the ground-truth method, allows to investigate the correctness of more complex scenes. The main disadvantage is

the loss of accuracy. If the reference algorithm has not yet been sufficiently tested for correctness, this method will not yield any meaningful insights.

One important issue is therefore to ease and simplify the test-process of rendering algorithms. A way to facilitate this is to support easy access to many different test scenes and verification techniques. Smits et al. [2] propose a database for test-scenes. They claimed, that such a database will help to lower the hurdles for the community to use test-scenes. It should increase the willingness to run tests on rendering algorithms. A software to be run as a publicly accessible database will be our partial solution to the previously described problems. The database should be held simple and easy to use. Ground-truth datasets like those of O. Clausen et al. [3] and Schregle et al. [4] and test-scenes which are based on comparison to other render-predictions, like those of Mardaljevic et al. [8] and Smits et al. [2] should all be supported by the database. An overview of state-of-the-art verification methods is shown in Section 2.1.

In this work, a prototype database was developed. It is a centralized place for test-scenes of different types. It is publicly available to everybody. The community can contribute with test-scene uploads, so others can download and use them. The concept in Section 3.1 also covers features for future work, including support for filter operations by various categories and tags. Some of the tags can be classified by properties and characteristics of the contained objects in the scene. An easy to use (graphical) user interface (UI) will allow for intuitive search and filter operations for specific scenes. It is shown in Section 3.1.3, that it is not trivial to classify scenes, but a community-driven approach like described in Section 2.3.1, with tagging-support [9] for each scene, is expected to automatically yield the best classification-system in the long run. More disadvantages of conventional approaches in contrast to community-driven platforms are shown in Section 3.1.1. As explained in Section 3.1.2, the community is thought to work with a reputation system similar to stackexchange-sites and all members will be able to edit and modify all uploaded scenes to get a consistent tagging-scheme throughout the database by the help of more experienced members [9]. The chosen frameworks for the implementation are listed in Section 3.1.4. Not all of the planned features from Section 3.1 were implemented. Actual implementation results are therefore documented in Section 3.2. The prototype is hosted on odpr.cg.tuwien.ac.at [10] and the source code can be found on gitlab.com [11].

CHAPTER 2

State of the Art

2.1 Verification of Rendering Algorithms

The test-procedure of rendering algorithms will not be part of this thesis, though, in order to get a better idea of the underlying intention of such datasets, two common testing-techniques shall be mentioned here:

- **Physically based testing:** Separate ground-truth models for each reflectance function need to be produced. In the case of O. Clausen et al. [3], this has been achieved for the diffuse reflectance function. Only a few variables can be investigated per study. O. Clausen et al. therefore varied the color property and the angle of the reflection-patches. They fixated other variables and therefore treated them as interferences. An overview of the known reflectance functions can be looked up in [12] (Chapter 2). The ground-truth dataset can be used to validate render-predictions by calculating mean errors between the original photograph and the generated render-outputs. [3, 4, 5] However, this is the most complex approach, where each dataset is very difficult to obtain and the scenes have to be held minimalistic. It is not possible to verify that rendering algorithms are reliable in terms of mixed light paths and more complex shapes and materials in a scene. It is also not possible to obtain ground-truth data for every possible reflectance- or transmission-behaviour known, at least not within an acceptable time. Figure 2.1 shows a simplified illustration of the comparison between the ground-truth photograph and the rendered prediction.

- **Perceptual based testing:** The comparison between the photograph of the real-world test-scene and its rendered correspondent is not based on physical spectral values, but can be done by standardized metrics based on human perception. This approach is easier to perform, though it has its disadvantages. The photographs

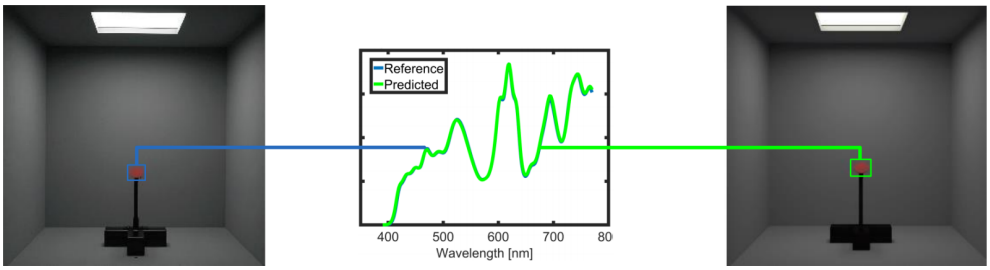

Figure 2.1: "Spectral radiance comparison of a real scene (left) with a predicted image based on our ground truth data (right)." Figure and description taken from O. Clausen et al. [3].

> do not have to be produced with expensive spectro-radiometers, but can be made with consumer-cameras. The test-scenes can be more complex, as the comparison relies on color-difference measurements instead of strict wavelength-observation. [1, 7, 13, 14]

More approaches including statistical verification of sampling algorithms and the ones listed above are discussed by Subr and Arvo [15] and in a comprehensive State-of-the-Art survey by Ulbricht et al. [1]

2.2 Taxonomy of Light Paths

Light sources and material types can be described easily. Light sources are normally divided into the following four types: Sun light, spot light, area light and point light. These are straightforward and can be parsed computationally from the 3D model's files. The materials and their reflection models are also encoded in those files. Also, they are defined through their reflectance functions (for example "diffuse", "specular" or "glossy"). However, when it comes to more general and complex optical phenomena or "effects", the author faces the problem of a proper and intuitive taxonomy. Some effects are widely known and used, for example "shadows" and "caustics". There is no name for every possible optical phenomenon yet.

However, there is an intuitive way to derive a code from the light paths in a scene. Figure 2.2 shows how this is done. This method was introduced by Heckbert et al. [16] and is called light path expression (LPE). The advantage is clearly the possibility to denote every possible optical phenomena simply by following the possible light paths. It is not even necessary to know how the "effect" actually looks. The path notation can be derived without this knowledge. There are always many possible paths at the same time. Fortunately, it is possible to unite all possible paths into a single and short regular expression. The three example paths in Figure 2.2 "LDDE", "LDE" and "LSDSE" can be denoted at once by the regular expression "L(D|S)*E". Another advantage is, that in theory, it is possible to let the computer generate these regular expressions for you.

You would need to modify a ray tracing algorithm so, that it outputs the letters for each hit on the corresponding objects. Then a script can find a matching regex for the list of light paths.

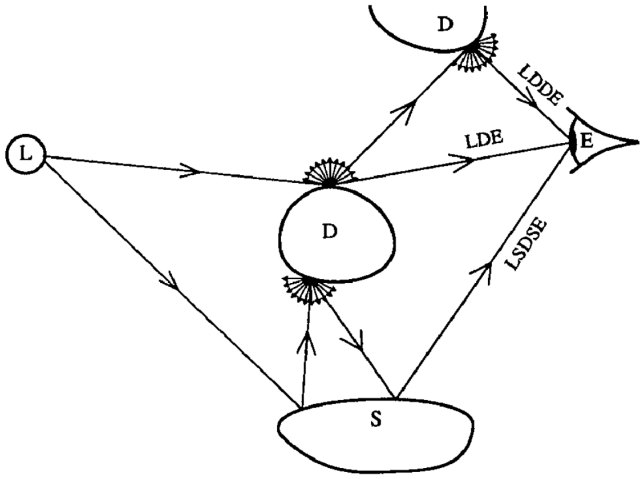

Figure 2.2: "Selected photon paths from light (L) to eye (E) by way of diffuse (D) and specular (S) surfaces. For simplicity, the surfaces shown are entirely diffuse or entirely specular; normally each surface would be a mixture." Figure and description taken from Heckbert et al. [16].

There are a few disadvantages though. The notation, especially with regular expressions, is easy to implement but not intuitive. Normally, people want to use uncomplicated schemes. The generation of the strings is amazingly simple. However, to read and understand already existing path notations is much more difficult. To simply read and understand fully pronounced names like "shadow" or "halo" would be less of a hurdle for humans. It is also not possible to include information about mixed materials. In reality, each surface is always some kind of mixed material. Every real object has diffuse and specular properties at the same time. It is indeed possible to describe such a mixture as a serial occurrence of the individual properties by "DS" or "SD". The exact percentage distribution can not be captured though.

2.3 Community-Driven Web-Applications

As the focus of this thesis lies on the ODPR-database, technical considerations as well as architecture- and design-considerations need to be addressed. In fact, there are already existing databases which are designed to host different 3D-Models, but they do not focus on the use case scenario to host test-data, but to share artistic or industrial models publicly, for example grabcad.com and blendswap.com.
Still, these two examples will be used to adopt basic design considerations and decisions.

2.3.1 State of the Art of Community-Driven Platforms

Fraternali et al. [9] made a very comprehensive overview of the design patterns and methods which are used and needed for community-driven web-platforms. They list key-concepts like members, items, activities, messages, permissions, rewards and reputation, which are all part of the basic design concept of ODPR and can also be seen on many platforms that inspired ODPR, for example grabcad.com, blendswap.com and stackexchange.com. The basic idea is, that users should become more active contributors, instead of just passively receiving content [9]. Various social and behavioral aspects are considered for effective design patterns of community-driven platforms [17].

Members are people who registered on the platform and act, communicate and change the state of the platform by their actions. They therefore also influence their own reputation.
Items are elements of interest. These represent the core of the application. In the example of YouTube, the elements of interest would be videos. In the case of ODPR, the items are the hosted test-scenes.
Activities include uploading, downloading, commenting, voting or rating, tagging, and further possible actions a member can do. These activities can be conditional, depending on permissions.
Permissions can depend on the role of a member:
Roles include *staff*, *moderator*, *admin*, *user* or even unregistered *guests*. While admin users have full access to almost everything in the application, guests only are granted restricted access.
Reputation points define the access level a user has. On stackexchange, users cannot vote or comment until they have enough reputation points to get access to these features. This has the effect of a spam-filter. New users not only tend to be inexperienced, they also might be fake profiles or robots. Restricting the access to features which allow users to change the state of the app is therefore crucial. Reputation points also have the purpose to encourage users to actively contribute to the application. This works because reputation points have the effect of rewards for specific actions. Each action can be rewarded with an increase or decrease of reputation points.

2.3.2 State of the Art of Security for user-provided content in Web-Applications

For the sake of simplicity for the prototype, the author omits more detailed security considerations. The provided default security mechanisms by Django, together with https (ssl), are considered to be sufficient. More considerations about better security for this platform are strongly advised for future development. Nermark [18] (chapter 2.3) provides an overview of the features and the architecture of Django, as well as security aspects. He also stated, that a running instance of Django should be updated by admins regularly to get important security updates for the latest discovered vulnerabilities.

CHAPTER 3

Methodological Approach

3.1 Concept

One initial and fundamental question is, what methods and design patterns should be used for the ODPR-database. Considerations about a minimal and straight-forward website which hosts test-scenes as a simple list, without any additional features except for the possibility to download scenes, are made in Section 3.1.1. It is shown, that such a minimalistic approach would not help to solve the main problem of this thesis (see Section 1).

Hence, community-driven design-patterns are examined in Section 3.1.2 in order to provide a real solution to that problem. There it is shown that most of the design considerations mentioned in Section 2.3.1 seem to fit well to the ODPR-database. Therefore, all possible and relevant use cases have been worked out. These use cases have been classified into mandatory use cases, important use cases and nice-to-have use cases. The mandatory use cases are needed for a minimum viable platform. The important use cases are not necessarily needed to be able to use the ODPR-database. They are highly appreciated and will enhance the user experience clearly. The nice-to-have use cases will enhance the user experience only in small levels. This classification was needed to be able to focus on the critical aspects of the prototype during the short time of its development. During this analysis, the features listed in Table 3.1 have been identified.

Because of their complexity, the filter criteria for optical phenomena are explained in their own Section 3.1.3.

Section 3.1.4 is about implementation details and considerations about frameworks and tools which are useful for the ODPR-prototype.

3.1.1 Problems and disadvantages with conventional Methods and Design Patterns

The minimal functionality that is necessary for such a database is the ability to download test scenes. This implies the need for a UI with some necessary elements. A simple design would consist of a single page. In this page, all test scenes would be listed in a simple, scroll-able list. Each scene in the list could be downloaded by clicking on a button. The test scenes would have to be stored in the database. Admins can do that without the need for any complex or appealing UI. For the most basic setup, user-based uploads would not be needed, since admins could directly add the new scenes in the backend.

However, the aim for the actual ODPR-prototype includes more features and flexibility. The goal is to lower the hurdles for the community to use and produce more test-scenes. Therefore, a simple list is not enough. The maintenance of such platforms is a lot of work for administrators. It is even more work to also maintain user-provided content manually. Admins would have a lot of repeating tasks to do. They would constantly need to manually update the database with new test-scenes. Also to look into each file would be an obligatory task. Those files have to be checked for illegal content. Users would have to download and open each test scene in order to see what the scene really contains. These are just a few aspects which contradict to the main goal of the ODPR-database. Some of them could introduce even more hurdles for the community to run tests.

3.1.2 Community-driven Approach

The goal is to distribute maintenance tasks as much as possible between the admins and users and to use automation wherever possible. The ODPR-database itself can automate some tasks. As a software, it can be expanded and adapted to its environment. Many tasks can still not be automated by software, though. At least not with low effort. So there is a second key aspect about the project. The development of the ODPR-platform should be possible in a short time by just one person. The check for spam or illegal content could be automated. Not in time by one developer, though. This lead to the second possible addressee of tasks: The community. Community-driven web applications are not a new concept. Preece [17] elaborated on social-behavioral aspects.

Community-driven aspects observed by Fraternali et al. [9] and Preece [17] have already been addressed in Section 2.3.1. ODPR makes use of them in the following manner: Scene uploads should not be performed by administrators. Users should be able to upload scenes on their own. This suggests to introduce **registered users** to make it easier to filter for spam and illegal content. Users can be blocked if they do not adhere to the site's rules and policies. On the other hand, they can be rewarded and marked as "trusted users", if they prove themselves over time by their actions. A "user"-table in the ODPR-database opens up a wide area of possible features in the social context.

With registered users it is also possible to implement **voting or rating** for scenes. Users can up- or downvote scenes. Upvotes have the purpose to grade a scene upon its quality and appropriateness. Downvotes tell the community that they can ignore or

avoid the item. Those meanings behind the votes address both, the community and the scene-maintainer. The scene-maintainer is the person who uploaded the scene. On the one hand, people are looking for high quality scenes for their tests. On the other hand, contributors can now gain experience and therefore successively increase the quality of their scenes. The votes act as feedback and as spam-filter. They can further be used by the search algorithm. Scenes in the overview-page can be sorted according to their votes. It would be possible to implement voting for anonymous unregistered guest-users. However, many disadvantages emerge from votes which can only be traced back to internet protocol (IP) addresses. Users and bots can use dynamic IP addresses or virtual private networks (VPNs) to circumvent barriers in the voting system. Furthermore, users should only be allowed one vote per scene. First, users can be restricted more easily to just one vote when registered. Second, unregistered and new users can completely be denied access to the voting feature. This also prevents users from registering multiple accounts to circumvent the voting barriers, because they still would need to obtain some reputation points before they are able to vote. Each new account will start with zero reputation points and is therefore not eligible to vote.

An important feature is the possibility for members to **report scenes**. Reports can be necessary for illegal, malicious or otherwise inappropriate or unsuitable content. Administrators of the platform will not check for such activities or content actively on a regular basis. The community will be inclined to use this feature, because it is in their interest to keep the ODPR-database free of spam. A report can simply be done by pressing the report-button which is visible on the detail-view of each scene.

The **detail-view** is a UI page, where a single scene can be viewed. It lists every aspect and information about the current scene. Contained files and images and metadata like the description, tags and statistics are shown in this page. Everybody can visit this page: Registered and unregistered users. Everybody can download the files from this page. This is in contrast to the **overview-page**, where many scenes are listed with minimal information per item. It is not possible to download anything from the overview-page. Some aspects like tags and description and a thumbnail are however visible in the overview-page. Interesting statistics which give an impression about the quality and content of the scenes are also condensed in the overview-page.

Comments are a nice way for the community to discuss scenes and give valuable feedback. A comment can only be made on the detail-view of a scene. Comments can also be voted. Users must adhere to social rules. Rudeness and bad language should be suppressed wherever possible. These would negatively impact the social cohesion and the public opinion about ODPR. Therefore, rebukes are possible through responding comments, votes and reports. Comments can be voted and reported just like scenes. The main purpose of comments is to give feedback about possible improvements of a scene.

Another important feature is the **deletion** of scenes. Not at least because it is required by European laws just as the report-button is as well. Only administrators, moderators and scene-maintainers are able to delete scenes, comments, requests and user accounts.

Moderators are volunteers with enough reputation to count as trustworthy. They will have more permissions than conventional users, but far less than administrators. Moderators can edit and delete scenes, comments, requests and answers. They can however not change the permissions of any user. They can also temporarily block and unblock user-accounts, but not delete them.

Requests are the proper way for registered users to ask in the community for specific scenes. Requests have the purpose to ask other people for help. Volunteer members might have the necessary skills, experience or tools to manufacture scenes the requesting person is not able to make on their own. An answer to a request will be uploaded as a new scene. The request-maintainer is the member who posted the request. This member will be able to mark one answer as the correct one to resolve the request. Just as for receiving votes, the reputation will increase for users who post answers which are marked as resolved.

In conclusion, the discussed features are planned to improve the social cohesion of the community and to reduce the hurdles to manufacture new test scenes and to finally test rendering algorithms. The platform can be seen as a social network where people connect. This allows for better mutual support than in individual research groups.

3.1.3 Filter Criteria

The platform is designed for a few thousand scenes. Therefore it is necessary to provide tools and mechanisms to filter scenes. A search bar can be used to search for specific scenes. They can be filtered by their names, their description, statistics like votes and download- or visit-counts. The most important filter, however, is made possible by the tagging system. For ODPR, four possible tags have been introduced: **Lights**, **Materials**, **Effects** and miscellaneous **Tags**. The latter are used for everything which does not fit into any of the other tag-categories.

In theory, those tags could be parsed from the uploaded files automatically. This is, however, not in the scope of this thesis. Thus, it is necessary to provide UI-elements for tag-input. Users can provide tags when they upload a new scene. If there are multiple elements of the same type in a scene, e.g. three point-lights, the corresponding field will only list the type once. The amount of them will not be covered by the tags. Whether a type of an element is included in the scene or not, is the focus of the tags.

Lights in general include the tag-values **Sun**-, **Spot**-, **Area**- and **Point**-light. In addition to that, color names could be added to this field. This allows to filter for scenes which only include specific light sources, or which include at least the light sources which are given in the search bar. Lights are especially of interest, because they make one major component for render predictions.

Materials include a wider range of possible values. In CG, real physical materials are approximated using mathematical models. Commonly, BRDFs and textures are used to represent the surface appearance of objects. In the case of ODPR, the UI-field for

Feature List			
ID	Name	Description	Importance
1	Scene Overview	The main page where all scenes are displayed as a list	mandatory
2	Scene Detail View	The details of a single selected scene, displayed on its own page	mandatory
3	Scene Upload	The possibility to upload new files and images to create a new test-scene on the website, which can then be viewed through features 1 and 2	mandatory
4	Scene Download	Users can download the data through the detail page	mandatory
5	Members	Users can register, login and logout	mandatory
6	Reports	Scenes can be reported by users	mandatory
7	Deletion	Scenes and accounts can be deleted by both, administrators and members (who uploaded the scene)	mandatory
8	Search and Filter	Search specific scenes by specific characteristics and filter criteria	important
9	Tags	Tags will enhance the classification of scenes in general and will yield better search results	important
10	Comments	Members can comment on uploaded scenes to suggest improvements	nice to have
11	Requests	Members can create request-posts to ask the community if they can create and upload scenes with specific, requested properties	nice to have
12	Votes	Scenes can be voted by members upon their quality	nice to have
13	Documentation	Members can contribute to the community by creating or editing documentation pages which describe how the site is supposed to be used, for what purpose it exists and which rules should be followed by each individual	nice to have
14	Reputation	Members can gain reputation points for uploading scenes of high quality and can lose reputation points for uploading scenes of low quality or of low suitability	nice to have
15	Stars	Members can mark scenes as favorites	nice to have

Table 3.1: Feature List

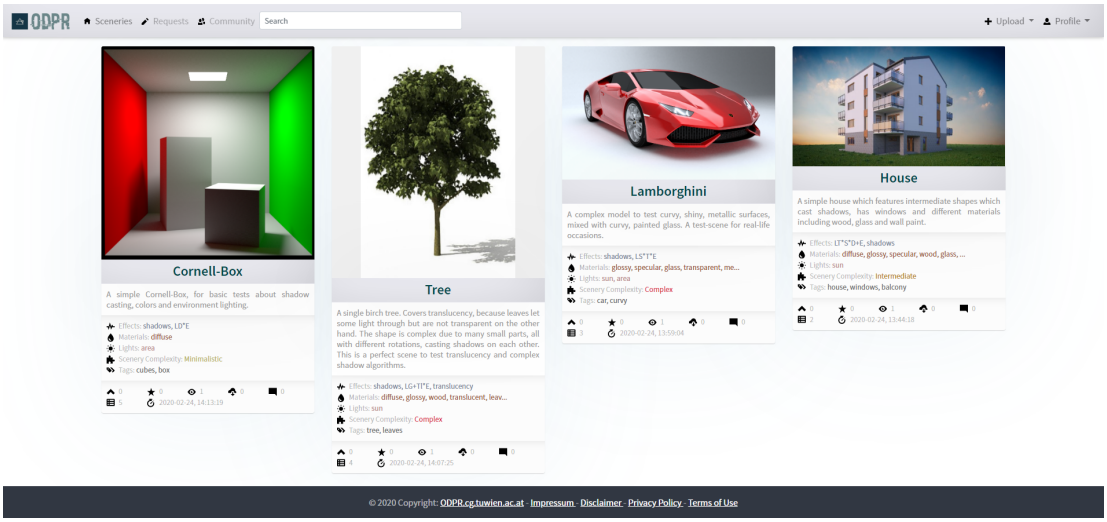

Figure 3.1: The overview-page of the ODPR-prototype. Four example-scenes have been uploaded to the database. Those are listed in this page, which is also the main page of the platform. More scenes are displayed in vertical direction, which can be reached through scrolling. Each so-called card displays the title, description, tags and statistics of a scene. The detail-view of a specific scene can be visited by clicking on the images of the cards. Furthermore, the header and the footer of the whole platform can be seen in this figure. The header currently consists of a logo, the three sections “Sceneries”, “Requests”, “Community”, and a search-bar. On the right side are either login- and register-buttons, or the two dropdown-menus “Upload” and “Profile”, if a user is already logged in. The upload-view (see Figure 3.3) can be reached through the dropdown-menu “Upload”. The footer consists of a copyright label, the domain of the site and links to impressum, disclaimer, privacy policy and terms of use. The example models in this screenshot are taken from free3d.com [19].

materials’ tags can contain reflection properties of the BRDFs, for example “diffuse” or “glossy”, but it can also contain certain texture names or the names of the real-world correspondents of the material, like “wood”, “metal”, “glass”, etc.

Effects are the most complex tags. As already explained in Section 2.2, there are not necessarily names for every possible optical phenomenon. Common names, like shadows, caustics, etc. can still be used as tags, but for more complex light paths, a more general and expressive alternative is necessary. One, where the name can be derived of the scene’s content instead of its render-output, because optical phenomena are only visible after they have been rendered. The light sources and their types, materials and their textures, however, are known before the scene is rendered. Hence, LPEs (see Section 2.2) fulfill this requirement. Therefore, in addition to conventional naming, the effects-tags will include regular expressions consisting of the symbols **L** for light, **E** for eye, **D** for diffuse and **S** for specular surfaces or units. More symbols can be added to denote more

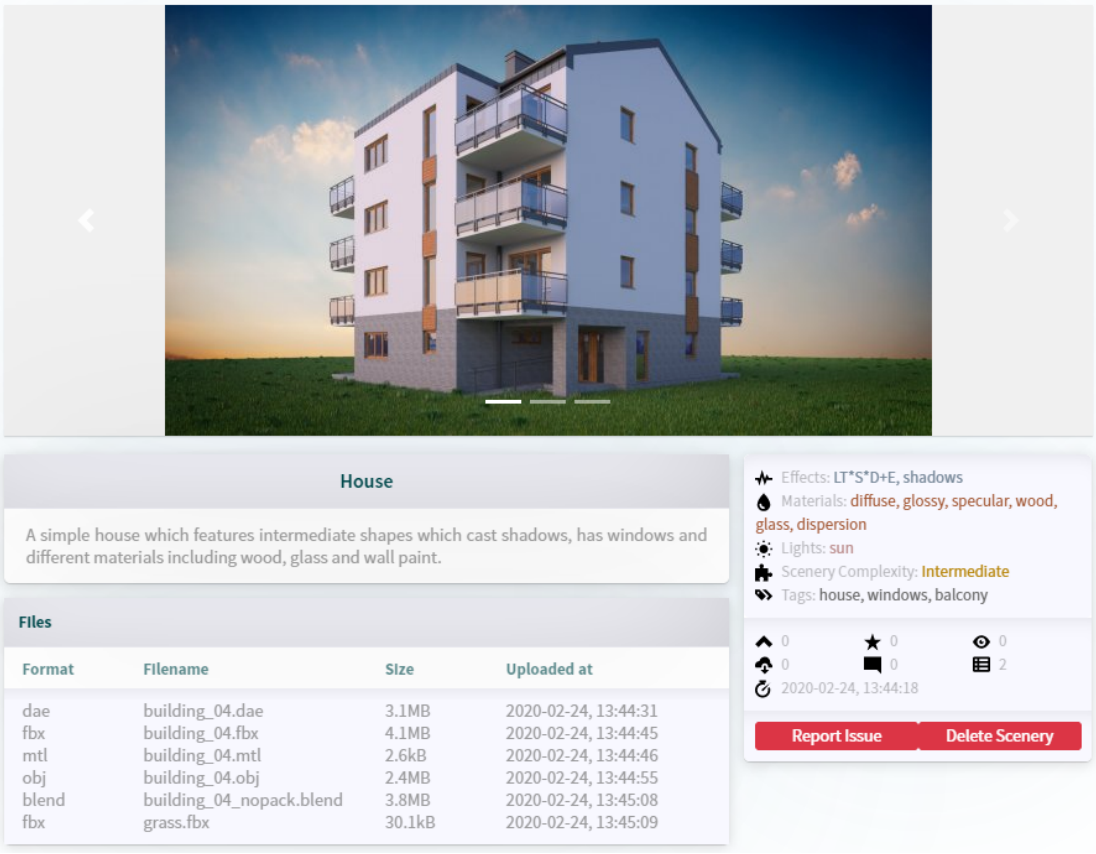

Figure 3.2: The detail-view of the ODPR-prototype. An example model has been uploaded to the database. Its rendered preview-image can be seen on top of the detail-view. In the middle-left section are the title and the description of the scene. Underneath are the raw 3D-model files, which can be downloaded by clicking on them. On the right side is a container with tags and statistics. The statistics include information about upvotes, stars, visits, downloads, comments, scene-ID and the upload-date. The report- and delete-button are located underneath the statistics. The delete-button is only visible for the scene-maintainer. The example model in this screenshot is taken from free3d.com [19].

properties including glossy materials (**G**), transmission (**T**) or volumes (**V**).

3.1.4 Choice of Frameworks

The next step was to choose the necessary tools and frameworks for the development and the app itself. Therefore, it was necessary to look at the features and their technical constraints and then choose one of several available frameworks for the categories: Back-end (BE), front-end (FE), deployment, database and development.

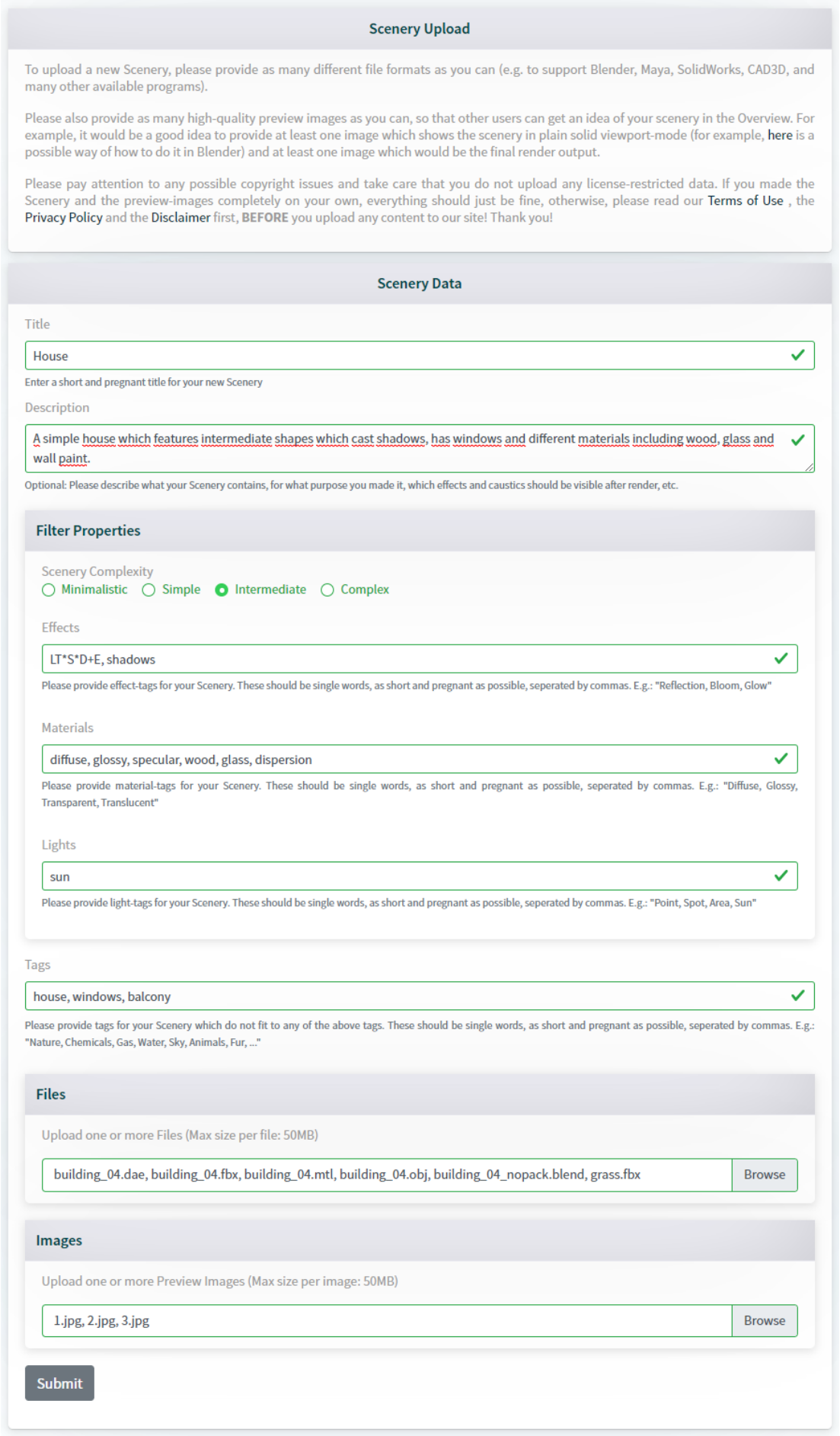

Figure 3.3: The upload-view is the most complex view.

(continued)

Figure 3.3: It starts on top with an introduction about the necessary files and elements, and states a strong note about the terms of use, disclaimer and privacy policy. Underneath, all necessary input fields can be seen. Those are the title, description, scene-complexity, four different types of tags, file- and image-browsers. Each section provides explanations and hints. The fields in this figure are filled with the example values for the scene shown in Figure 3.2. When all necessary input fields are filled with data, the submit-button at the bottom becomes valid and a click on it will start the upload of the files (see Figure 3.4 and Figure 3.5).

Figure 3.4: After the upload-button was clicked, the files are uploaded successively. Feedback about the progress is provided through the symbols which can be seen on the right side. If a file could not be uploaded, a red “x” will appear. If a file is successfully saved on the database, the green hook can be seen. The current active upload is shown by the rotating blue circle. The rest of the files is marked as “queued”.

Available frameworks have not been analyzed extensively, because the only criteria for their choice was whether they provide the required functionality or not. Therefore, Django was chosen for the BE, Vue was chosen for the FE, docker for the deployment and postgresql for the database. Nginx was chosen to serve the deployed website. Webpack and Node.js® (especially npm) were used during the development of the FE and pip during the development of the BE.

3.2 Results

3.2.1 UI-Design and implemented Features

Not everything from Section 3.1 is included in the ODPR-prototype. This section provides an overview about which features from Table 3.1 have been added and which are still to add in future work.

Figure 3.1 shows the main page of ODPR. It implements the features **Scene Overview** and **Members** from Table 3.1. Although UI-elements for the features **Search and Filter**, **Tags** and **Requests** are already visible, they are not implemented and either

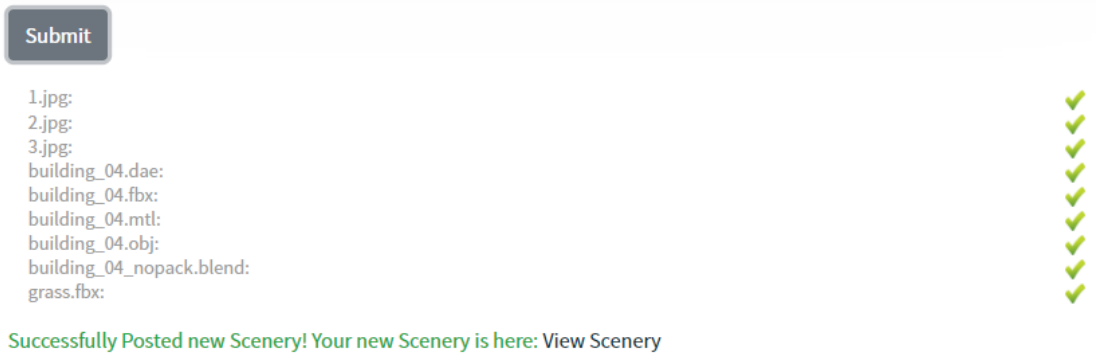

Figure 3.5: When the upload of a scene has finished, all files should have green hooks on the right side and a success-message can be seen at the bottom. The message provides a link to the detail-view of the newly uploaded scene.

greyed-out or not functional.

The **Scene Detail View**, **Scene Download**, **Reports** and **Deletion** from Table 3.1 can be seen in Figure 3.2. Missing are the features **Comments**, **Votes** and **Stars**, which would mainly be active on this page. Their implementation, however, has already started by adding the necessary database tables to the Django-models.

Metadata like tags can already be added to each new scene (see Figure 3.3 which shows the upload-view), but they are not used by the search-bar yet. Thus, feature (**Tags**) is not finished. Also, some aspects can be improved in future work: Already existing tags could be suggested to users in popup-menus when tags are typed in the corresponding input fields.

Visual progress-feedback during scene-uploads helps users to be patient especially for large files and slow internet connections (see Figure 3.4 and Figure 3.5). Error messages are also provided in this screen when something goes wrong. Users are therefore able to distinguish between successful scene-uploads or failed uploads, so they can start another attempt in the latter case.

The features **Documentation** and **Reputation** are also not included in the prototype. The former should provide more in-depth explanations and tutorials for the community about rules and netiquette in general and in particular about the tagging system and LPEs. The latter would open a wide range of possible features which have already been explained in Section 2.3.1.

3.2.2 Interplay of the chosen Frameworks

To support all necessary features, a combination of different software components (as mentioned in Section 3.1.4) was chosen. Due to the amount of technologies in use, their interplay will be illustrated on the following example:

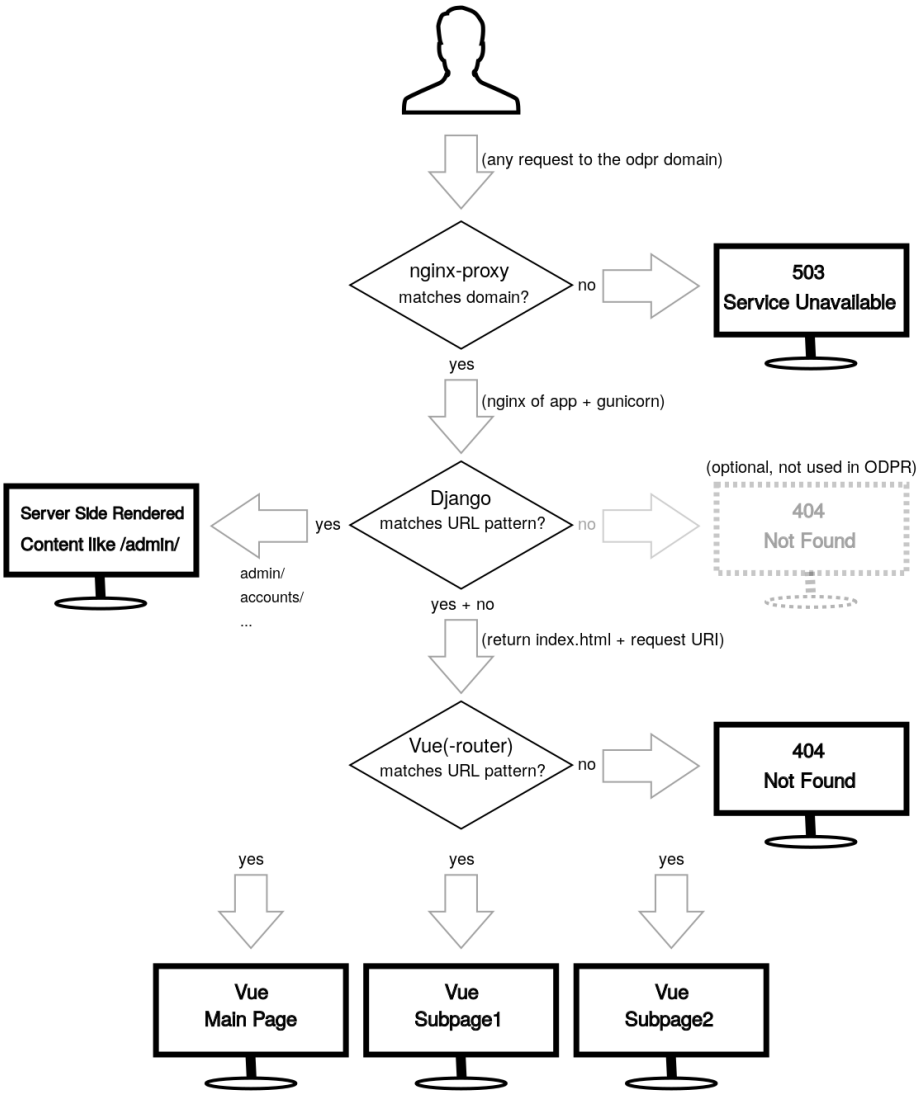

Figure 3.6: The illustration shows the different stages where the different responsibilities of the three services (nginx, Django, Vue) become active and how they respond to a request. A fourth and optional service, nginx-proxy, receives requests from the web and tries to match apps which registered their domain to it, which allows to serve multiple apps with different domains on the same machine (see also Figure 3.7). If the app is running, the request will be forwarded through the app-specific nginx-container to its Django-BE. Django will match the URL and will either return a response directly or forward the request to the Vue-FE. Vue is the last instance which will always respond with a result, be it an error or a regular page.

Consider user Bob, who wants to visit the website odpr.cg.tuwien.ac.at and clicks on that link (illustrated with the top silhoutte and its down-pointing arrow in Figure 3.6). His browser will send a GET-request to the host machine to the service which listens to the ports 80 and 443 (which are the standard ports for http and https). On that machine, ODPR has previously been deployed in three seperate docker containers: One for nginx, one for Django and the last one for the database (represented by the two bottom diamond shapes and the "nginx of app"-arrow in Figure 3.6). The nginx-container exposes the two ports 80 and 443 to the host and to the public web, as those two ports are already exposed by the host machine as well. The http-request coming from Bob's browser will therefore be received by the nginx-container and nginx will resolve the URL in the request according to the configuration of nginx, which is also included in the nginx-container. The configuration will tell nginx, inter alia, which files to return on which requests, how to return them (compressed or uncompressed), file-size-limits on uploads, redirect-instructions from http to https, and finally the forwarding of requests to the second container where the Django BE will handle them further.

Requests forwarded by nginx to Django will be resolved according to metadata like the URL, body, request-type (GET, PUT, POST, DELETE, and more), and will return responses accordingly. The first request of Bob will match the Django URL "/", which represents the root of the BE. "/" is mapped to a so-called "View", which returns a file named "index.html" for this specific request.

That "index.html" has previously been generated during the build-process in webpack and is a small file with minimal content, so it can be served as fast as possible even if Bob has a very slow internet connection. The rest of the content of the app is just referenced by this file, but itself split up into many other small files, so-called chunks, which are then automatically requested by Bob's browser as soon as that client has parsed the "index.html" and recognized, that there are more files to fetch.

Those chunks are now fetched through their own requests to the nginx-service, but this time, the URL will match a so-called "static-files" configuration, which are, like the name says, just static files, which will (almost) never need to change (except for app-updates, hence the "almost"), hence they can be returned by nginx directly without the detour to Django and then back to nginx until they would finally be served to the client Bob.

Now, as Bob's client has finally downloaded all of those static-files, which are the source code of the FE, a mechanism will inject code into the "index.html". This is done because the "index.html" is the only file which is ever loaded in the client as a page, hence the name Single Page Application (SPA). Every sub-page on the website which Bob visits will be dynamically injected or replaced by the next one, directly in that opened "index.html" file.

Next, Bob sees the fully loaded site, but there is still something going on in the background of ODPR: The start-page of ODPR is supposed to display a list of test-scenes to provide a nice overview for users and visitors (see Figure 3.1). Scenes, however, are not part of the static-files, because scenes are not hard-coded into the app's source code, they are

rather dynamically provided by users, thus “dynamic files”. They have to be fetched separately every time a user (re-)loads the page, because they could have changed since the last (re-)load, even if that was just seconds ago. A so-called application programming interface (API) has been implemented for this purpose, and Vue utilizes a http-client (in this case axios) to send similar http-requests like Bob’s client did earlier when Bob opened the site for the first time. This time, however, the request will not be visible to Bob directly through the browser’s address bar. It will be issued in the background by the mentioned http-client instead and will equally arrive at the nginx-service of the hosting machine.

The nginx-service again tries to figure out what to do with the requested URL and finds no matching entry except for the Django service. The request will therefore be detoured to Django, which then again will try to match the request’s URL against its own URL-tables (the so-called API) and will handle the request accordingly. In this case, Bob’s client requested a list of all scenes for the overview-page. The app is configured so, that it will do a lookup in the database and will return eight scenes at a time, a so-called “paginated” response. The response goes back to nginx which will send it back to Bob’s client.

Bob’s client, in this case the http-client inside the Vue-app, receives the list of scenes, passes it to the caller (a Vue-component), and the component will inject the received scene-data into the “index.html” to display them properly.

Other use-case scenarios, like scene-uploads, login, logout, register, scene-details work basically in the same way. Some of the described events are not shown in Figure 3.6.

3.2.3 Deployment

Figure 3.7 shows an overview of the different parts of the app, in the context of its build-, test- and production-environments. To be able to deliver an app in time, continuous integration (CI) and continuous deployment (CD) have proven to be a good choice for such projects. Without such tools, the development and deployment become tedious and error prone. Different machines and different operating systems (OSs) could introduce unpredictable errors or make the deployment impossible.

Hence, the app is delivered as a docker image bundle. Those can be run as OS-independent, isolated containers on every machine which has docker-support. A gitlab-runner builds the docker images, runs tests on their containers and deploys the app if the tests passed and if the new version should be deployed.

Figure 3.7 shows also, how nginx-proxy can distribute incoming requests to multiple apps, if needed.

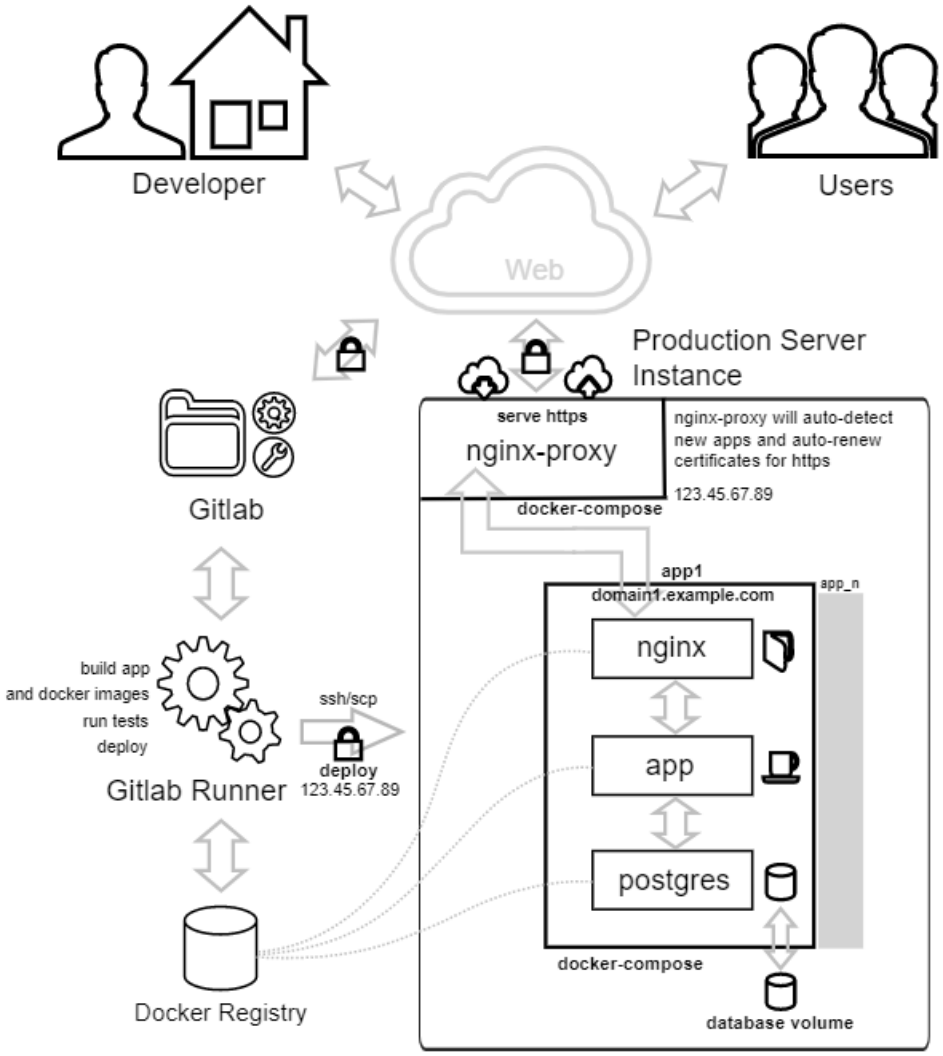

Figure 3.7: This figure shows the deployed app on the production server, as well as its development process and how it is automatically built and tested on a gitlab CI/CD instance. The app is built and deployed with three docker images: An image for nginx, another for the ODPR-BE and the last one for the PostgreSQL instance. Each app is started with docker-compose. Other apps can be run on the same host. The optional docker-compose setup "nginx-proxy" will serve the correct app depending on the requested URL. Inside each app, the initial request from outside, is forwarded by nginx-proxy to the nginx instance of the app, which forwards the request further to the BE. The BE resolves the requested path and responds with a proper result. If necessary, it will communicate with the postgres-database.

CHAPTER 4

Conclusion

One key to the verification of rendering algorithms is the access to test-methods and test-data. The other key is to motivate the community to actually use them and run more tests. A prototype for a community-driven web-platform was designed and developed as part of this thesis. The so-called Open Database for Physically-based Rendering provides a central access-point for various test-scenes to the CG community. Anticipated intentions of users and the resulting requirements were analyzed and a feature-list (Table 3.1) was worked out. Much of the attention went to the tagging-system. Especially the capture of optical phenomena seemed to be difficult at first. However, the notation with LPEs constituted a simple and powerful solution.

A lot of features are still to be implemented. Some hypothetical feature ideas can be considered for future work. The former include all of the "important" and "nice-to-have" features from Table 3.1. They could not be addressed during the implementation of the prototype. The implementation of some were started in the BE, but they were not finished in either or both: the FE and BE. Others still have to be started.

The latter include the automatic analysis of the uploaded test-scenes for possible tag-values. The three fields for **Lights**-, **Materials**- and **Effects**-tags can be filled with values, in addition to user-provided ones, by the inspection of the test-scene's files. Other possible features for future development are expected to emerge from the feedback of the community.

List of Figures

List of Tables

Glossary

Database If used without "ODPR"-prefix, except for occurrences in chapter 1, an SQL-like database instance is meant. Technical aspects and the data itself and how it is stored and retrieved are in the focus of this term. 5, 8, 13, 15, 18, 19

Detail-View The page on the platform where an individual scene is shown like in Figure 3.2. All information about a scene is displayed in this screen, including UI-elements for downloading and reporting this particular scene. 9, 12, 13, 16, 22, 23

Diffuse Reflection Light is being scattered equally in all directions, when it hits a surface. [16] 3–5, 12, 22, 25

Effects Optical phenomena like shadows or caustics are covered by the term "effects" in this thesis. This term is not only used for optical phenomena. It is also mainly used for the name of a tagging field in the UI of the ODPR-database. The notation for effect-tags is explained in 2.2. ..... 4, 10, 12, 21

Glossy Reflection Light is being scattered with mixed: diffuse and specular reflection behaviour, when it hits a surface. 4, 12, 13

ODPR-Database If used with "ODPR"-prefix and in Section 1, the whole platform "ODPR" is meant. The design behind UI-elements and the community-driven aspects are in the foreground. Technical aspects are addressed without the prefix. 5, 7–9

Overview-Page The main page of the platform where the most relevant aspects of multiple scenes are displayed simultaneously. The summarized information of four example scenes can be seen in Figure 3.1. ..... 9, 12, 19, 22

Reflectance Function A reflectance function describes the interaction of light with matter. Due to the complexity of the General Reflectance Function (GRF), simplified models have been created. Those focus on just a few of the 16 variables of the GRF and are meant to be used for special use cases. Those simplifications include the following reflectance functions: BRDF, BDTF, BRTTF, BSDF, BSSRDF, BTDF and BTF [12] (Chapter 2) 3, 4

Specular Reflection Light is being scattered in only a small cone of directions, when it hits a surface. If the cone(s) have a positive finite angle, it is called “rough specular”. If that angle is exactly zero, it is called “ideal specular”. [16] . 4, 5, 12, 22, 25

Upload-View The page on the platform where new scenes can be uploaded. Model files, preview-images and metadata is added to the scene in this step, like it can be seen in Figure 3.3. ..... 12, 14, 16, 22, 23

Acronyms

API	Application Programming Interface	pp. 18 f.
BDTF	Bidirectional Dynamic Texture Function	p. 25
BE	Back-end	pp. 13, 16, 18 f., 21, 23
BRDF	Bidirectional Reflectance Distribution Function	pp. vii, ix, 10, 25
BRTTF	Bidirectional Reflectance Transmittance Texture Function	p. 25
BSDF	Bidirectional Scattering Distribution Function	p. 25
BSSRDF	Bidirectional Surface Scattering Reflectance Distribution Function	p. 25
BTDF	Bidirectional Transmittance Distribution Function	p. 25
BTF	Bidirectional Texture Function	p. 25
CD	Continuous Deployment	p. 19
CG	Computer Graphics	pp. 1, 10, 21
CI	Continuous Integration	p. 19
FE	Front-end	pp. 13, 16, 18, 21, 23
GRF	General Reflectance Function	p. 25
IP	Internet Protocol	p. 8
LPE	Light Path Expression	pp. 4, 12, 16, 21
ODPR	Open Database for Physically-based Rendering	pp. vii, ix, 5 ff., 15 f., 18 f., 21 ff., 25
OS	Operating System	p. 19
SPA	Single Page Application	p. 18
UI	(Graphical) User Interface	pp. 2, 7, 9 f., 15, 25
VPN	Virtual Private Network	p. 8

Bibliography

[1] Christiane Ulbricht, Alexander Wilkie, and Werner Purgathofer. Verification of physically based rendering algorithms. *Computer Graphics Forum*, 25(2):237–255, 2006.

[2] Brian Smits and Henrik Wann Jensen. Global illumination test scenes. Technical report, University of Utah, 2000.

[3] O. Clausen, R. Marroquim, and A. Fuhrmann. Acquisition and validation of spectral ground truth data for predictive rendering of rough surfaces. *Computer Graphics Forum*, 37(4):1–12, 2018.

[4] Roland Schregle and Jan Wienold. Physical validation of global illumination methods: Measurement and error analysis. *Computer Graphics Forum*, 23(4):761–781, 2004.

[5] Victor A Debelov and Dmitri S Kozlov. Rendering of translucent objects, verification and validation of algorithms. 2012.

[6] Raphaël Labayrade and Vincent Launay. Test cases to assess the accuracy of spectral light transport software. *International IBPSA Building Simulation Conference*, 2011.

[7] Ann McNamara. Exploring visual and automatic measures of perceptual fidelity in real and simulated imagery. *ACM Trans. Appl. Percept.*, 3(3):217–238, July 2006.

[8] J Mardaljevic. The bre-idmp dataset: a new benchmark for the validation of illuminance prediction techniques. *Transactions of the Illuminating Engineering Society*, 33(2):117–134, 2001.

[9] Piero Fraternali, Massimo Tisi, Matteo Silva, and Lorenzo Frattini. Building community-based web applications with a model-driven approach and design pattern. *Handbook of research on Web*, 2(3.0), 2008.

[10] Odpr - open database for physically-based rendering. `https://odpr.cg.tuwien.ac.at/sceneries`. Accessed: 2020-03-03.

[11] Odpr - source code (gitlab project). `https://gitlab.com/electrocnic/odpr`. Accessed: 2020-03-03.

[12] Michal Haindl and Jiri Filip. *Visual texture: Accurate material appearance measurement, representation and modeling.* Springer Science & Business Media, 2013.

[13] G. B. Meneghel and M. L. Netto. A comparison of global illumination methods using perceptual quality metrics. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 33–40, Aug 2015.

[14] Jenny Benois-Pineau and Patrick Le Callet. *Visual Content Indexing and Retrieval with Psycho-Visual Models.* Springer, 2017.

[15] Kartic Subr and James Arvo. Statistical hypothesis testing for assessing monte carlo estimators: Applications to image synthesis. In *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, pages 106–115. IEEE, 2007.

[16] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *SIGGRAPH Comput. Graph.*, 24(4):145–154, September 1990.

[17] Jenny Preece. *Online communities: Designing usability and supporting socialbilty.* John Wiley & Sons, Inc., 2000.

[18] Magnus Nermark. Automatic notification and execution of security updates in the django web framework. Master's thesis, NTNU, 2018.

[19] Free3d. `https://free3d.com/`. Accessed: 2020-02-28.