*Todo list

MSc in Computer Science 2017-18

Project Dissertation

Project Dissertation title: Reflectance Transformation Imaging

Term and year of submission: Trinity Term 2018

Candidate Name: Johannes Bernhard Goslar

Title of Degree the dissertation is being submitted under: MSc in Computer Science

Abstract

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Acknowledgements

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background in Applications

Todo Text:
Background in Applications

1.2 Background in Computer Science

Todo Text:
Background in Computer Science

2 Related Work

Todo Text:
Related work intro

2.1 RTI Theory and Workflows

Todo Text:
Workflow Comparisions

2.2 Fileformats

The most comprehensive overview on the current state of the art is done by the American library of congress as part of its Digital preservation effort, with the sections on the ptm[2] and rti[3] formats. The current PTM specification by Malzbender and Gelb[8].

Todo Text:
File formats comparison

Todo Diagramm:
Size tables/graphes of ptm/rti/btf(.zip)

Todo Text:
Streaming architectures

2.3 RTI Viewers

Todo Text:
Viewer Comparision

Todo Text:
No extensible architecture

Todo Text:
No real open source (email before or one file sources)

2.4 Camera Theory

Todo Text:
Camera Theory

3 Methodology

Exploratory piece of work

3.1 Requirements

Todo Text:
Requirements Analysis, informal discussion

3.2 Architectural Design

Todo Text:
Architecture Picks

4 Requirements and Design

4.1 Requirements

Distilling these, I arrived at the following functional requirements, which are logically grouped into fileformat/support and viewer.

For the fileformat:

1. Support for the PTM[2] fileformat.
2. Support for the RTI[3] fileformat.
3. Conversion of the formats above into a unified format.
4. Extended metadata support.
5. Support for high resolutions.
6. Support for higher bitdepths per pixel than the 8 of PTM/RTI.
7. Easy exchange between multiple researchers.

For the viewer component:

8. Runnable on all major operating systems and/or web browsers.
9. Lightning Controls.
10. Quick navigation functionality.

Continuing the enumeration of the functional requirements, following functional requirements were extracted:

11. Free Software, the implementation should be available for everyone to change and distribute.
12. Easy on-boarding of new developers, either scientists in a research context or students in an education context.
13. Good developer experience.
14. Adequate performance, at least keeping up with current implementations.
15. Easy installation for researchers.
16. “Web”-Based.
17. Instant reactiveness.

18. Reasonable file sizes for instant transfer/viewing.

4.2 State-Driven

Todo Text:
State-Driven

4.3 Plugins

Todo Text:
Plugins

4.4 Rendering Stack

Todo Text:
Rendering Stack

4.5 Workflow

Todo Diagramm:
Workflow comparison

json

Todo Text:
File import/export

4.6 Novelties

Todo Text:
Novelties Design

5 Implementation

5.1 Overview

This section explains the current implementation of the developed tool set, it is primarily targeted to fulfill the dissertation's requirements. But is also aiming to be helpful for users wanting to understand the underlying systems and prepare them for potentially joining the development effort. Abridged code extracts are used as of their state for thesis submission, while the main principles will hold, later readers are asked to please consult the actual source code if any discrepancies arise or reexport the document. First the main libraries are shortly explained in their relevance to the program, second the largely abstract plugin architecture is shown, third the main plugins are presented and last the delivery processes to the end users are described.

All implementation files are contained and delivered inside a single git repository, which is freely available online: `https://github.com/ksjogo/oxrti`. All following file paths are relative to that repository's root. All future development will be immediately available there and the current compiled software version is always fed automatically from it into the hosted version at `https://oxrtimaster.azurewebsites.net/api/azurestatic`.

5.2 Libraries

TypeScript: The official header line of TypeScript show some points why it was picked for this project: "TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. Any browser. Any host. Any OS. Open source."[15] Which fits requirements 11, 8. Whereas plain JavaScript would have allowed slightly easier initial on-boarding and maybe easier immediate code 'hacks', TypeScript will provide better stability in the long run and a quite improved developer experience (requirement 13) in the long run. With the full typed hook system (compare section 5.9) it ensures that a compiled plugin will not have runtime type problems, reducing the amount of switching between code editor and the running software. The whole project is setup in a way to fully embrace editor tooling, Visual Studio Code[16] and Emacs[6] are the 'officially' tested editors of the project. Code is recommended as it will support all developer features out of the box. The installation of the tslint[13] plugin[14] is recommended to keep a consistent code style, which is configured within the *tslint.json* file. Most importantly TypeScript adds type declarations (and inference) to JavaScript, e.g.:

```
const thing = function (times: number, other: (index: number) => boolean) { ... }
```

would define *thing* as a function, taking a numbers as first argument and another function (taking a number as first parameter and returning a boolean) as second argument. The other most used TypeScript features inside the codebase are Classes[1], Decorators[4] and Generics[5], which will be discussed at their first appearance inside the code samples.

React: The two main points on React's official website are "Declarative" and "Component Based"[11], which is best shown by an extended example from their website, which exemplifies multiple patterns found through the oxrti implementation. The most important concept is the jump from having a stateful HTML document, which the JavaScript code is manipulating directly, e.g.:

```
document.getElementById('gsr').innerHTML="<p>You shouldn't do this</p>"
```

Which is diametric to requirements 12 and 13 as it would require developers to manually keep track of all data cross-references (e.g. the pan values having to automatically adapt to the current zoom level). A declarative approach instead allows much better and easier implemented reactiveness and better performance (requirements 14 and 17) as the necessary changes can be track and components be updated selectively.

```
// a class represents a single component
class Timer extends React.Component {
  // the parent component can pass on props to it
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    //  the state is updated and the component is automatically rerendered
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
```

```
  }

  // called after the component was created/added to the
  ↪ browser window
  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  // called before the component will be deleted/removed
  ↪ from the browser window
  componentWillUnmount() {
    clearInterval(this.interval);
  }

  // the actual rendering code
  // html can be directly embedded into react components
  // {} blocks will be evaluated when the render method
  ↪ is called
  // which will happen any time the props or its internal
  ↪ states updates
  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

// mountNode is a reference to a DOM Node
// the component will be mounted inside that node
ReactDOM.render(<Timer />, mountNode);
```

In conjunction with mobx and TypeScript no classes are used for React components though, but instead Stateless Functional Components ('SFCs'[7]). These SFCs are plain functions, only depending on their passed properties:

```
function SomeComponent(props: any) {
 return <p>{props.first} {props.first}</p>
}
```

This component could then be used by:

```
 <SomeComponent first="Hello" second="World"/>
```

This component systems allows the plugins to define some components and then ‘link’ them into the program via the hook system, which will be explored later.

mobx: Its main tagline is “Simple, scalable state management”[9]. An introducing overview is shown in Figure 1. Broadly speaking MobX introduces observable objects. Instead of mentioned DOM handling or property passing inside React trees, components can just retrieve their values from the observable objects and will be automatically refreshed if the read values change. This for example makes the implementation of the QuickPan plugin extremely easy, as it can just read the zoom, pan, etc. values of the other plugins and will automatically receive all updates without any further manual observation handling.

mobx-state-tree: “Central in MST (mobx-state-tree) is the concept of a living tree. The tree consists of mutable, but strictly protected objects”[10] This allows the implementation to have one shared state tree which can be used to safely access all data. All nodes inside the state tree are MobX observables. A simple tree with plain MST would look like this:

```
// define a model type
const Todo = types
 .model("Todo", {
  // state of every model
  title: types.string,
  done: false
 })
 .actions(self => ({
  //methods bounds to the current model instance
  toggle() {
   self.done = !self.done
  }
 }))
// create a tree root, with a property todos
const Store = types.model("Store", {
    todos: types.array(Todo)
})
```

This syntax was deemed to convoluted, as it is a lot more complex

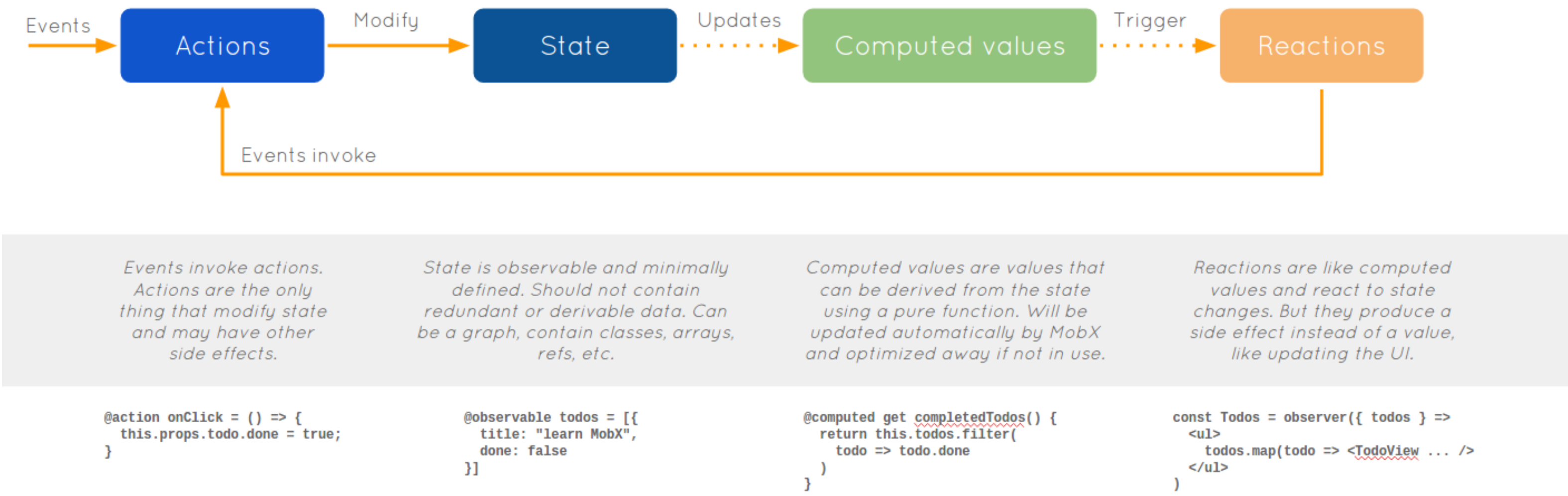

Figure 1: Taken from Weststrate[9]. Actions in the oxrti context are most often initially user actions, which are then calling into plugins to change the state. The state is mostly encapsulated on a plugin basis with usage of the mobx-state-tree library, which also encapsulates most computed values. Reactions are most often the previously discussed React components.

than standard JavaScript/TypeScript ES6 classes as shown in the React description above and thus being in conflict with requirement 12.

classy-mst:

gl-react:

> **Todo Text:**
> gl-react

webpack:

> **Todo Text:**
> webpack

[12]

electron:

> **Todo Text:**
> electron

misc:

> **Todo Text:**
> misc

[9]

5.3 Plugin architectures

> **Todo Text:**
> Plugin architectures

```
function murks() : number{}
```

5.4 BTF File Format

This section describes the BTF file format. The aim of this file format is to provide a generic container for BTF data to be specified using a variety of common formats. Files shall have the `.btf.zip` extension.

5.4.1 File Structure

A BTF file is a ZIP file containing the following:

- A **manifest** file in JSON format, named `manifest.json`. The manifest contains all information about the BRDF/BSDF model being used, including the names for the available **channels** (e.g. `R`, `G` and `B` for the 3-channel RGB), the names of the necessary **coefficients** (e.g. bi-quadratic coefficients) and the **image file format** for each channel.

- A single folder named `data`, with sub-folders having names in 1-to-1 correspondence with the channels specified in the manifest.
- Within each channel folder, greyscale image files having names in 1-to-1 correspondence with the coefficients specified in the manifest, each in the image file format specified in the manifest for the corresponding channel. For example, if one is working with RGB format (3-channels named `R`, `G` and `B`) in the PTM model (five coefficients `a2`, `b2`, `a1`, `b1` and `c`,specifying a bi-quadratic) using 16-bit greyscale bitmaps, the file `/data/B/a2.bmp` is the texture encoding the `a2` coefficient for the blue channel of each point in texture space.
- The datafiles are all in reversed scanline order (meaning from bottom to top), to keep aligned with the original PTM format and allow easier loading into WebGL.

5.4.2 Manifest

The manifest for the BTF file format is a JSON file with root dictionary. The `root` element has two mandatory child elements: one named `data`, and one named `name` with the option of additional child elements (with different names) left open to future extensions of the format.

- The `name` element is a string with a name of the contained object.
- The `data` element has for entries, named `width`, `height`, `channels` and `channel-model`. The `width` and `height` attributes have values in the positive integers describing the dimensions of the BTDF. The `channel-model` attribute has value a non-empty alphanumeric string uniquely identifying the BRDF/BSDF colour model used by the BTF file (see Options section below). The `channels` element has an arbitrary amout of named `channel` entries, according to the `channel-model`. * Additionally the `data` element has one untyped entry named `formatExtra`, where format implementation specific data can be stored.
- Each `channel` has an `coefficents` child consisting of an arbitrary number of `coefficent` entries, as well as one `coefficient-model` attribute. The `coefficient-model` attribute has value a non-empty alphanumeric string uniquely identifying the BRDF/BSDF approxima-

tion model used by the BTF file (see Options section below). * Each `coefficient` element has one attribute: `format`. The `format` attribute has value a non-empty alphanumeric string uniquely identifying the image file format used to store the channel values (see Options section below).

5.4.3 Textures

Each image file `/data/CHAN/COEFF.EXT` has the same dimensions specified by the `width` and `height` attributes of the `data` element in the manifest, and is encoded in the greyscale image file format specified by the `format` attribute of the unique `coefficient` element with attribute `name` taking the value `COEFF` (the extension `.EXT` is ignored). The colour value of a pixel `(u,v)` in the image is the value for coefficient `COEFF` of channel `CHAN` in the BRDF/BSDF for point `(u,v)`, according to the model jointly specified by the values of the attribute `model` for element `channels` (colour model) and the attribute `model` for element `coefficients` (approximation model).

5.4.4 Options

At present, the following values are defined for attribute `channel-model` of element `channels`.

- `RGB`: the 3-channel RGB colour model, with channels named `R`, `G` and `B`. This colour model is currently under implementation. * `LRGB`: the 4-channel LRGB colour model, with channels named `L`, `R`, `G` and `B`. This colour model is currently under implementation.
- `SPECTRAL`: the spectral radiance model, with an arbitrary non-zero number of channels named either all by wavelength (format `---nm`, with `---` an arbitrary non-zero number) or all by frequency format `---Hz`, with `---` an arbitrary non-zero number. This colour model is planned for future implementation.

At present, the following values are defined for attribute `model` of element coefficients, where the ending character `*` is to be replaced by an arbitrary number greater than or equal to 1.

- `flat`: flat approximation model (no dependence on light position). This approximation model is currently under implementation.

- `RTIpoly*`: order-`*` polynomial approximation model for RTI (single view-point BRDF). This approximation model is currently under implementation.

- `RTIharmonic*`: order-`*` hemispherical harmonic approximation model for RTI (single view-point BRDF). This approximation model is currently under implementation.

- `BRDFpoly*`: order-`*` polynomial approximation model for BRDFs. This approximation model is planned for future implementation.

- `BRDFharmonic*`: order-`*` hemispherical harmonic approximation model BRDFs. This approximation model is planned for future implementation.

- `BSDFpoly*`: order-`*` polynomial approximation model for BSDFs. This approximation model is planned for future implementation. * `BSDFharmonic*`: order-`*` spherical harmonic approximation model for BSDFs. This approximation model is planned for future implementation.

At present, the following values are defined for attribute `format` of elements tagged `coefficient`, where the ending character `*` is the bit-depth, to be replaced by an allowed positive multiple of 8.

- `BMP*`: greyscale BMP file format with the specified bit-depth (8, 16, 24 or 32). Support for this format is currently under implementation.

- `PNG*`: PNG file format encoding the specified bit-depth (8, 16, 24, 32, 48 or 64). Support for this format is currently under implementation. Different PNG colour options are used to support different bit-depths: * `Grayscale` with 8-bit/channel to encode 8-bit bit-depth. * `Grayscale` with 16-bit/channel to encode 16-bit bit-depth. * `Truecolor` with 8-bit/channel to encode 24-bit bit-depth. * `Truecolor and alpha` with 8-bit/channel to encode 32-bit bit-depth.

- `Truecolor` with 16-bit/channel to encode 48-bit bit-depth.

- `Truecolor and alpha` with 16-bit/channel to encode 64-bit bit-depth.

5.5 Loader

5.6 State Management

Todo Text:
state management

Todo Text:
state import/export

Todo Diagramm:
redux

Todo Diagramm:
mobx actions

5.7 Components

Todo Text:
single component units

5.8 Renderer Stack

Todo Text:
base rendering nodes

Todo Diagramm:
stacked components

Todo Diagramm:
effects

5.8.1 Texture Loading

5.9 Hooks

The hook system allows stable and prioritized interactions between the different plugins. All available hooks are declared inside the Hook.tsx file, which offers 3 different types of hooks:

```
// Hooks are sorted in descending priority order in their respective `HookManager`
export type HookBase = { priority?: number }

// Generic single component hook, usually used for rendering a dynamic list of components
export type ComponentHook<P = PluginComponentType> = HookBase & { component: P }

// Generic single component hook, usually used for notifications
export type FunctionHook<P = (...args: any[]) => any> = HookBase & { func: P }

// Generic hook config, requiring more work at the consumer side
export type ConfigHook<P = any> = HookBase & P

// union of all hooks to allow for manual hook distinction
export type UnknownHook = ComponentHook & FunctionHook & ConfigHook

// object of named hooks
type Hooks<P> = { [key: string]: P }

// collection of unknown hooks
export type UnknownHooks = Hooks<UnknownHook>

// hook configuration inside plugins: 1-Hookname->*-LocalName->1-HookConfig
export type HookConfig = { [P in keyof HookTypes]: Hooks<HookTypes[P]> }

```

```
// all hooknames
export type HookName = keyof HookConfig

// map one hookname to its type
export type HookType<P extends HookName> = HookTypes[P]

// list of hooknames inside hook collection T, having hooktype U
type LimitedHooks<T, U> = ({ [P in keyof T]: T[P] extends U ? P : never })[keyof T]

// limit hookname parameters to a type conforming subset, e.g. LimitedHook<ComponentHook>
export type LimitedHook<P> = LimitedHooks<HookConfig, Hooks<P>>
```

These types are used to first declare single hook types (which will be discussed within the plugins consuming them) and then construct the whole hook configuration tree for all plugins:

```
type HookTypes = {
  ActionBar?: ConfigHook<ActionBar>,
  AfterPluginLoads?: FunctionHook,
  AppView?: ComponentHook,
...
}
```

5.10 Plugins

Todo Text:
Plugins API

5.10.1 Base Plugin

Todo Text:
Base Plugin

5.10.2 BaseTheme Plugin

Todo Text:
Basetheme Plugin

5.10.3 RedTheme Plugin

5.10.4 TabView Plugin

```
type Tab = {
    content: PluginComponentType
    tab: TabProps,
    padding?: number,
    beforeFocusGain?: () => Promise<void>,
    afterFocusGain?: () => Promise<void>,
    beforeFocusLose?: () => Promise<void>,
    afterFocusLose?: () => Promise<void>,
}

type ActionBar = {
    onClick: () => void,
    title: string,
    enabled: () => boolean,
    tooltip?: string,
}

type ViewerTabFocus = {
    beforeGain?: () => void,
    beforeLose?: () => void,
}

type ScreenshotMeta = {
    key: string,
    fullshot?: () => (string | number)[] | string | number,
    snapshot?: () => (string | number)[] | string | number,
}

type ViewerFileAction = {
    tooltip: string,
```

```
    text: string,
    action: () => Promise<void>,
}
```

Todo Text:
TabView Plugin

5.10.5 SingleView Plugin

Todo Text:
SingleView Plugin

5.10.6 Converter Plugin

Todo Text:
Converter Plugin

5.10.7 PTMConverter Plugin

Todo Text:
PTMConverter Plugin

5.10.8 Renderer Plugin

```
type BaseNodeConfig = {
    channelModel: ChannelModel,
    node: PluginComponentType<BaseNodeProps>,
}

type RendererNode = {
    component: PluginComponentType,
    inversePoint?: (point: Point) => Point,
}

type MouseConfig = {
    listener: MouseListener,
```

```
    mouseLeft?: () => void,
}
```

Todo Text:
Renderer Plugin

Todo Text:
Base Node

Todo Text:
WebGL texture packing

5.10.9 PTM Renderer Plugin

Todo Text:
PTM Renderer Plugin

Todo Text:
Dynamic Shaders

Todo Text:
RGB vs LRGB

5.10.10 Light Control Plugin

Todo Text:
Light Control Plugin

5.10.11 Rotation Plugin

Todo Text:
Rotation Plugin

5.10.12 Zoom Plugin

Todo Text:
Zoom Plugin

5.10.13 QuickPan Plugin

Todo Text:
Zoom Plugin

5.10.14 Paint Plugin

Todo Text:
Zoom Plugin

5.10.15 Import Export Plugin

Todo Text:
Automatic Import Export

5.11 Applications

Todo Text:
Other related graphics

Todo Text:
Applications

5.11.1 Standalone Website

Todo Text:
Standalone Website

5.11.2 Embeddable

Todo Text:
Embeddable

5.11.3 Electron

Todo Text:
Electron App deliverable

6 Results

6.1 Featureset

Todo Text:
Featureset Comparison

Todo Diagramm:
Screeshots

6.2 Performance

Todo Text:
Performance

6.3 Testing

Todo Text:
Testing

Todo Text:
Shader Interpolation

Todo Text:
Image comparison

6.4 Rollouts and Deployments

Todo Text:
Rollout

Todo Text:
Non-Tech deployment

o

7 Discussion

7.1 Community Onboarding

Todo Text:
Community Onboarding

7.2 Novelties

Todo Text:
Novelties results

7.3 Future Work

The future work can be split into two parts. Improvements of the current system, including better performance and bug fixes, and further extensions with new functionality.

Todo Text:
Future Work

8 Conclusion

Todo Text:
Conclusion

References

[1] *Classes · TypeScript*. URL: `https://www.typescriptlang.org/docs/handbook/classes.html` (visited on 08/20/2018).

[2] Library of Congress. *Polynomial Texture Map (PTM) File Format*. June 14, 2018. URL: `https://www.loc.gov/preservation/digital/formats//fdd/fdd000487.shtml` (visited on 08/10/2018).

[3] Library of Congress. *Reflectance Transformation Imaging (RTI) File Format*. June 9, 2018. URL: `https://www.loc.gov/preservation/digital/formats//fdd/fdd000486.shtml#notes` (visited on 08/10/2018).

[4] *Decorators · TypeScript*. URL: `https://www.typescriptlang.org/docs/handbook/decorators.html` (visited on 08/20/2018).

[5] *Generics · TypeScript*. URL: `https://www.typescriptlang.org/docs/handbook/generics.html` (visited on 08/20/2018).

[6] *GNU Emacs - GNU Project*. URL: `https://www.gnu.org/software/emacs/` (visited on 08/17/2018).

[7] Takahiro Ethan Ikeuchi. *React Stateless Functional Component with TypeScript*. Medium. Apr. 5, 2017. URL: `https://medium.com/@ethan_ikt/react-stateless-functional-component-with-typescript-ce5043466011` (visited on 08/20/2018).

[8] Tom Malzbender and Dan Gelb. "Polynomial Texture Map (.ptm) File Format". In: (), p. 6.

[9] *mobx: Simple, scalable state management*. Aug. 13, 2018. URL: `https://github.com/mobxjs/mobx` (visited on 08/13/2018).

[10] *mobx-state-tree: Model Driven State Management*. Aug. 20, 2018. URL: `https://github.com/mobxjs/mobx-state-tree` (visited on 08/21/2018).

[11] *React - A JavaScript library for building user interfaces*. URL: `https://reactjs.org/index.html` (visited on 08/13/2018).

[12] Gaëtan Renaudeau. *gl-react – React library to write and compose WebGL shaders*. Aug. 13, 2018. URL: `https://github.com/gre/gl-react` (visited on 08/13/2018).

[13] *TSLint*. URL: `https://palantir.github.io/tslint/` (visited on 08/17/2018).

[14] *TSLint - Visual Studio Marketplace*. URL: `https://marketplace.visualstudio.com/items?itemName=eg2.tslint` (visited on 08/17/2018).

[15] *TypeScript is a superset of JavaScript that compiles to clean JavaScript output*. Aug. 13, 2018. URL: `https://github.com/Microsoft/TypeScript` (visited on 08/13/2018).

[16] *Visual Studio Code - Code Editing. Redefined.* URL: http://code.visualstudio.com/ (visited on 08/17/2018).