DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Smartphone-assisted Virtual Reality using Ubi-Interact

Michael Lohr

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Smartphone-assisted Virtual Reality using Ubi-Interact

Smartphone-gestützte Virtuelle Realität mit Ubi-Interact

Author:	Michael Lohr
Supervisor:	Prof. Gudrun Johanna Klinker, Ph.D.
Advisor:	Sandro Weber, M.Sc.
Submission Date:	October 15, 2019

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, October 15, 2019 Michael Lohr

Acknowledgments

Abstract

Contents

Abbreviations

API Application Programming Interface

Protobuf Google Protocol Buffers

IMU Inertial Measurement Unit

JS JavaScript

UBII UBI-Interact

UID Unique Identifier

VR Virtual Reality

3D Three-dimensional

1 Introduction

Hi, this is my thesis, and I’m going to be the introduction.

2 Implementation

2.1 Ubi-Interact

UBI-Interact (UBII) is a framework for distributed applications, which enables to connect all kinds of different devices together. A centralized server is used to manage the system in a local network. It is currently developed and maintained by Sandro Weber, who is also the advisor for this thesis. The abstraction into devices, topics and interactions allows to decouple the implementation of a software from device specific environments.

2.1.1 Architecture

The main components of the UBII framework are:

Clients describe a basic network participant. For every client registered in the system, exists one network socket adress. So clients are an abstraction of a physical network device. They are defined by a Unique Identifier (UID).

Devices can be registered by clients. A device is an abstraction for a virtual device, which groups different input and output devices together. It is defined by a UID and a list of components.

Components contain the topic name, message format for input/output devices and wether it publishes input or receives output data. A data source for such an input device, could be any sensor for example a button or the camera. Data output examples for input devices are lamps and displays.

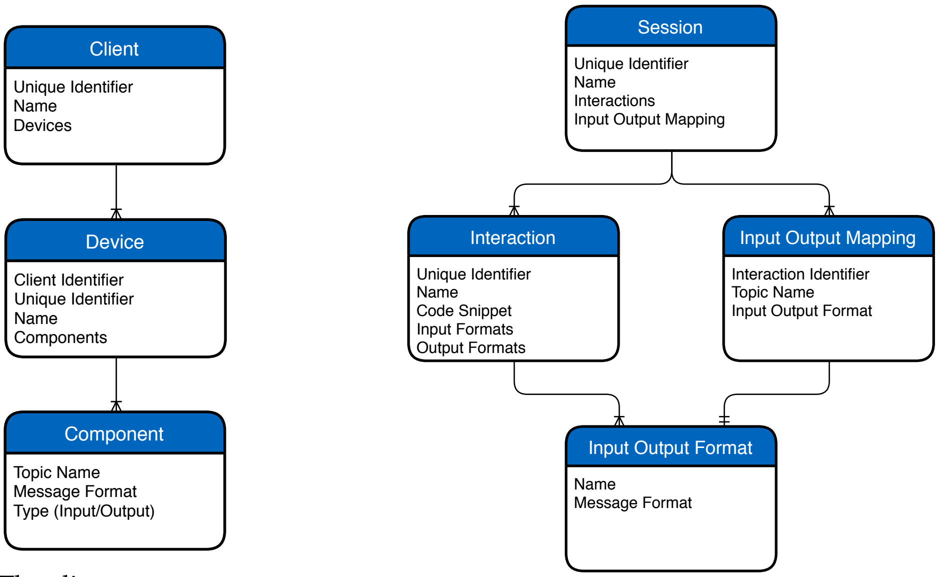
Message Formats are strings which identify a certain data format. Most common data types are available. For example *Vector4×4* (a four by four matrix), *Vector2* (a two-dimensional vector) or *boolean* (a binary value) are built in.

Topics are data channels which are addressed by a name. Clients can publish messages to topics, which are registered by a device. Clients are also able to receive messages, after subscribing to a topic. Such messages (also called "topic data") are formatted as JSON[1]-string, whose structure is defined by the device.

Sessions operate on the server, but can be specified by the client. They are defined by a UID as well as a list of interactions and **input/output mappings**. The mappings are defined by a message format and topic name.

Interactions are reactive components. They operate on topics and are defined by a source code snippet[2]. Interactions are executed in a fixed interval on the UBII server. They can subscribe to topics and use the received topic data as input, given an input/output mapping description. The output of the interaction is published into another topic. It is also possible to keep data to use in future executions (persistent state).

The figure 2.1 visualizes the relationships of the different components.



(a) The client components.

(b) The session components.

Figure 2.1: Relationships of the core components in an entity relationship diagram.

[1]JSON is a standardized data exchange format, that uses human-readable text. It is often used for web-based data communication.

[2]Currently only JavaScript is supported as a script language.

2.1.2 Interactions

An powerful but optional core feature of UBII are so called interactions. As explained in the component overview (see 2.1.1), they are reactive components, which operate on topics and regularly execute given code snippets (processing functions) on the UBII server. Interactions are isolated components, which just depend on topic data and nothing else. This abstraction introduces the possibility to reuse logic in other applications in similar context. The data flow from a device to the interaction is visualized in the figure 2.2.

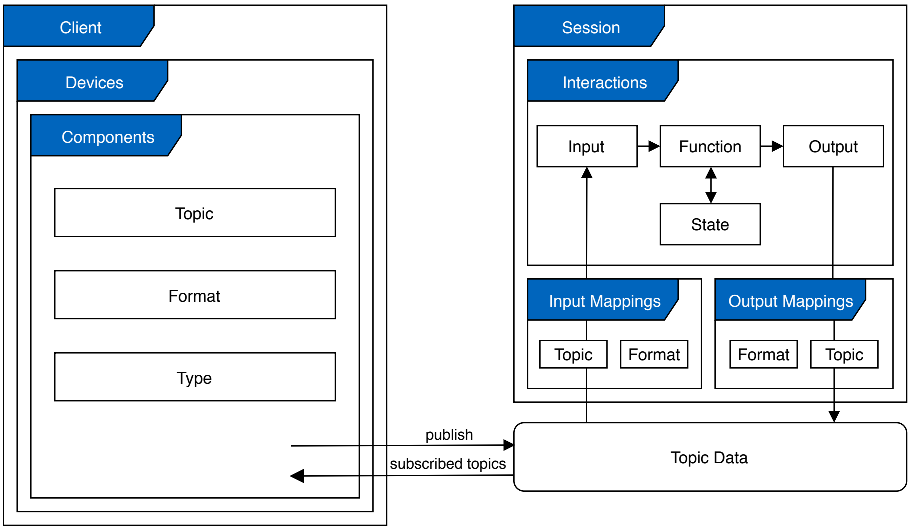


Figure 2.2: Interaction processing overview. This graphic gives a rough overview of the dataflow when using an interaction. Conceptualized with the help of Sandro Weber.

Interactions should be designed generalized, so that they are easy to reuse. They can be used to discretize data, converting data to other formats or just to outsource some logic from the application. Concrete examples include detecting button presses, transforming coordinates and evaluate data. An example implementation which detects movements can be seen in figure 2.3.

They are also useful, if you want to connect two topics with different formats. Imagine an application, which consumes a rotation given in euler angles in radians. But some devices publish euler angles in degrees. We could implement an interaction, which takes euler angles in degrees from one topic and publishes euler angles in radians to another one, which then is used by the application.

```
// detect intentional movement by comparing the current position with a previous one
function (inputs, outputs, state) {
  const threshold = 0.05;

  if (state.position) {
    const vector = {
      x: inputs.position.x - state.lastPosition.x,
      y: inputs.position.y - state.lastPosition.y,
    };

    const squaredDistance = Math.pow(vector.x, 2) + Math.pow(vector.y, 2);

    outputs.moved = squaredDistance < threshold;
  } else {
    outputs.moved = true;
  }

  state.lastPosition = inputs.position;
}
```

Figure 2.3: An example for an interaction written in JavaScript. This interaction calculates the squared distance of two points. One of the points comes from the input, while the other one is stored in the state. To achieve this, the euclidean vector norm of the subtraction of both vectors without the square root is calculated and compared with a threshold constant. The result is then written into the output as a boolean data type.

The code snippet has to define a function, which accepts three parameters: `inputs` is a collection of values, which contains values which were published into a topic. The topic which was used, is defined by the input mappings of the session. `outputs` is an empty collection, where values can be added. Those values are then published into a topic, defined by the output mappings of the session. `state` stores a persistent collection of values, which can be used in later executions of the same interaction.

2.2 Technology Stack

Since most of the existing software for UBII was written in JavaScript (JS)[1] using a fully web-based architecture, I decided to adapt this approach. This has one major

[1]JS is a just-in-time compiled scripting language, widely used in web technology. It is a dynamic prototype-based language, which supports object-orientated programming.

advantage: platform independence. Most modern devices can run web-based software, which means they can also run the software. Also the software is served by a web server, which means the user does not have to install the software onto his device.

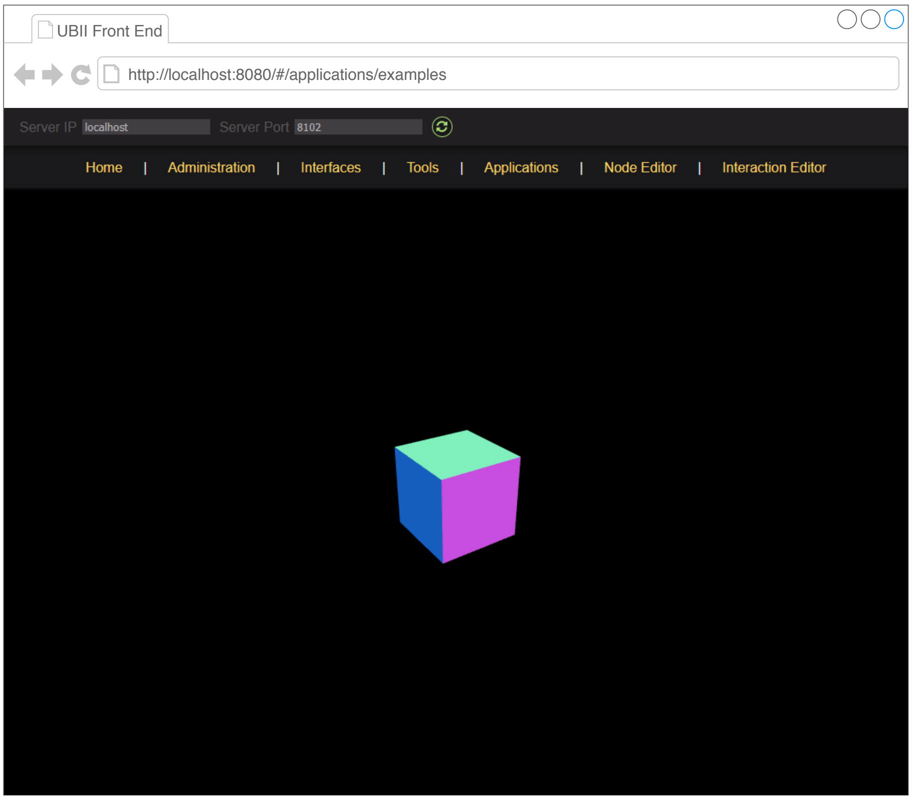


Figure 2.4: A screenshot of the UBII front end rendering a 3D cube.

A web interface with some UBII content (the UBII front end), demos and debugging tools was already written[1], so I included my demos in this interface as well. The technology stack of the front end was built with the following technologies:

Web APIs are Application Programming Interfaces (APIs) available in modern web browsers to provide access to functionality or data outside the browser. The WebAPI is an additional layer of abstraction on top of the operating system ones. While this has the advantage that the API is the same on every device, this also prevents the access to the raw sensor data[2]. In this thesis the WebVR API, which enables to render to external Virtual Reality (VR) headsets and the

[1]The front end was developed by Sandro Weber, Daniel Dyrda and me.
[2]The specification is available on www.w3c.github.io/deviceorientation

device orientation API, which gives access to the data of the Inertial Measurement Unit (IMU)[1] of a device, are used.

Vue.js [2] is a modern open source JavaScript web framework[3] [Eva19]. Being released in 2014 and developed by Evan You, it is a relatively young framework [**Koetsier.2016**]. But it quickly gained traction and is quite popular now [**Koetsier.2016**]. Modules like Vue.js itself, Vue.js plugins and other JavaScript libraries are managed using the package manager npm [4].

Three.js [5] is a lightweight open source library which utilizes WebGL to render Three-dimensional (3D) computer graphics [Ric19]. It can be used to render scenes to the display as well as to a VR headset using WebVR. This high-level library comes with a lot of features, similar to a game engine, like scenes, effects, lights, animation, geometrie and much more.

UBII Client is an JavaScript client for the UBII system. It abstracts the protocol and provides high-level functions to register devices as well as send and receive topic data. The UBII system uses Google Protocol Buffers (Protobuf)[6] to serialize the data.

Figure 2.4 displays an test view, which uses Vue.js to manage the views and Three.js to render a cube.

2.3 Smart Device

The "Smart Device" is a part of the UBII front end. Because it is web-based, only data which is available through the **Web APIs** can be obtained. Since it was not designed for a specific use case, it is thought as general purpose or testing device. Only touch positions, touch events, orientation and acceleration is sent to different topics using the **UBII Client**. For more specific scenarios, the smart device can not be used and a custom interface has to be implemented. For the experiments in this thesis though, the

[1] An IMU is an electronic component which is part of most smart phones and allows to measure force, angular rate and magnetic field.

[2] Vue.js: Website: www.vuejs.org, Source code: www.github.com/vuejs/vue

[3] A web framework is a software framework which provides a standard way to build web applications. It comes with tools and libraries to automate and make the development of web applications easier.

[4] "NPM" stands for "Node Package Manager" and is also used in the UBII server itself. Website: www.npmjs.com

[5] Three.js: Website: www.threejs.org, Source code: www.github.com/mrdoob/three.js

[6] Protobuf is a mechanism to serialize data. The data is defined in a platform-neutral language, which compiles as library to all commonly used programming languages [Goo19]. Website: www.developers.google.com/protocol-buffers/

smart device client was sufficient, after implementing some improvements.

The data which is published, is also displayed on the screen for debugging purposes. It is possible to set the view in full screen mode, to prevent unintentional interactions with control elements of the web browser or the operating system. Since the reference system for the orientation is fixed to the earth [Dev19, Chapter 4.1], a calibration system was implemented. With the press of the "Calibrate" button, the device is calibrated to the new orientation.

2.3.1 Topic Data

The orientation is provided by the **Web API** through the `DeviceOrientation` event. The orientation is described by three euler angles named `alpha`, `beta` and `gamma`, as seen in figure 2.5. While `alpha` returns values in the range $[0, 360)$, `beta` only returns the range $[-180, 180)$ and `gamma` $[-90, 90)$ [Dev19, Chapter 4.1]. This limitation entails that no full orientation tracking is possible with this event.

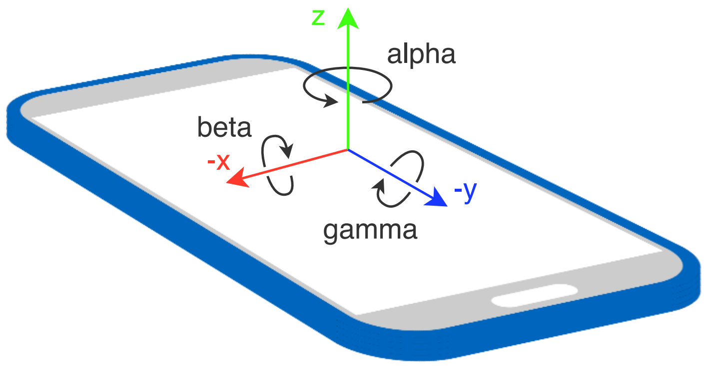


Figure 2.5: The specification of the orientation values visualized. The x and y axes are inverted for the sake of clarity.

The **Web API** also provides the `MotionEvent`, which returns multiple vectors, one being the acceleration with the gravity (`accelerationIncludingGravity`). Since the gravity vector always points down, this vector can be used as a reference vector. Together with the values from the `DeviceOrientation` event, the full orientation can be derived. The resulting orientation, then has to be smoothed, because the acceleration vector uses the raw IMU acceleration output.

The data from the `DeviceOrientation` already provides all three euler angles and is smoothed. Implementing the same for the data from the `MotionEvent`, would be outside of the scope of this thesis. Because of this consideration, the smart device uses the `DeviceOrientation` event data.

The touch position on the display, is normalized to a range from 0 to 1. This removes the influence of the display resolution and size. Touch events for start and stop touching are sent on a different topics. The acceleration is also sent to a topic, but is not used in this thesis.

2.3.2 UBII Device Definition

The smart device is registered as a device in the UBII network. The definition in JS can be seen in figure 2.6. The structure of a UBII device was described in Chapter 2.1.1.

```
const ubiiDevice = {
  name: 'web-interface-smart-device',
  components: [
    {
      topic: clientId + '/web-interface-smart-device/touch_position',
      messageFormat: 'ubii.dataStructure.Vector2',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    },
    {
      topic: clientId + '/web-interface-smart-device/orientation',
      messageFormat: 'ubii.dataStructure.Vector3',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    },
    {
      topic: clientId + '/web-interface-smart-device/linear_acceleration',
      messageFormat: 'ubii.dataStructure.Vector3',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    },
    {
      topic: clientId + '/web-interface-smart-device/touch_events',
      messageFormat: 'ubii.dataStructure.TouchEvent',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    }
  ]
};
```

Figure 2.6: The smart device definition in JavaScript. It is defined by a name and a list of components. The structure of a UBII device is further described in Chapter 2.1.1.

A device and all topics must be registered for each client, because it should be possible to read the data from different devices. This allows for using multiple devices at the same time, so that they can be differentiated in interactions. If the topic names would not include the `clientId`, each connected device would publish to the same topic, which would make the data unusable.

```
syntax = "proto3";
package ubii.dataStructure;

import "proto/topicData/topicDataRecord/dataStructure/vector2.proto";

enum ButtonEventType {
  UP = 0;
  DOWN = 1;
}

message TouchEvent {
  ubii.dataStructure.ButtonEventType type = 1;
  ubii.dataStructure.Vector2 position = 2;
}
```

Figure 2.7: The definition of the touch event, sent by the smart device client when a user touches or releases the touch screen. It is defined by a position and wether the touch pad was touched or released.

At the time of creating the experiments presented later, the component for the touch events was not implemented yet. The touch position is published multiple times per second, but only sends the current position of the first touch on the smartphone display. Using this data, it is not possible to detect wether the display was just touched or released. A new topic using the `Boolean`-type could have been used, but the position has to be obtained from the touch position topic. To remove this dependency on the other topic, the new type `TouchEvent` was implemented. The Protobuf definition can be seen in figure 2.7. It contains the two-dimensional position and the binary type `ButtonEventType`, which can be reused in other events, too. `ButtonEventType` is an enumerated type which defines wether the touch interface was just touched or released.

2.4 Experiments

Three main experiments were implemented to show how the smartphone can help with common interactions when using VR software. To achieve consistency amongst all experiments in terms of look and functionality, a parent class was implemented. The parent class uses modules like a sky module, I implemented, to set up a basic scene. Also the connection to the UBII server is handled.

2.4.1 Model Viewer

This experiment implements an 3D model viewer. When a smart device connects to the UBII network, a new instance of a model is created. This model can be rotated, by rotating the smart phone running the smart device client. The rotation of the phone, published on the orientation topic is applied to the model.

3 Introduction

3.1 Section

3.1.1 Cite Tests

Citation test [Afo+17] vs [Afo+17].
More [pre Afo+17, post]
Also possible[2]

Acronym test VR cool.
And a second time VR awesome.

3.1.2 Draw Test

3.1.3 Other Tests

See Table 3.1, Figure 3.2, Figure 3.3, Figure 3.4.

Table 3.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

[2] Afo+17.

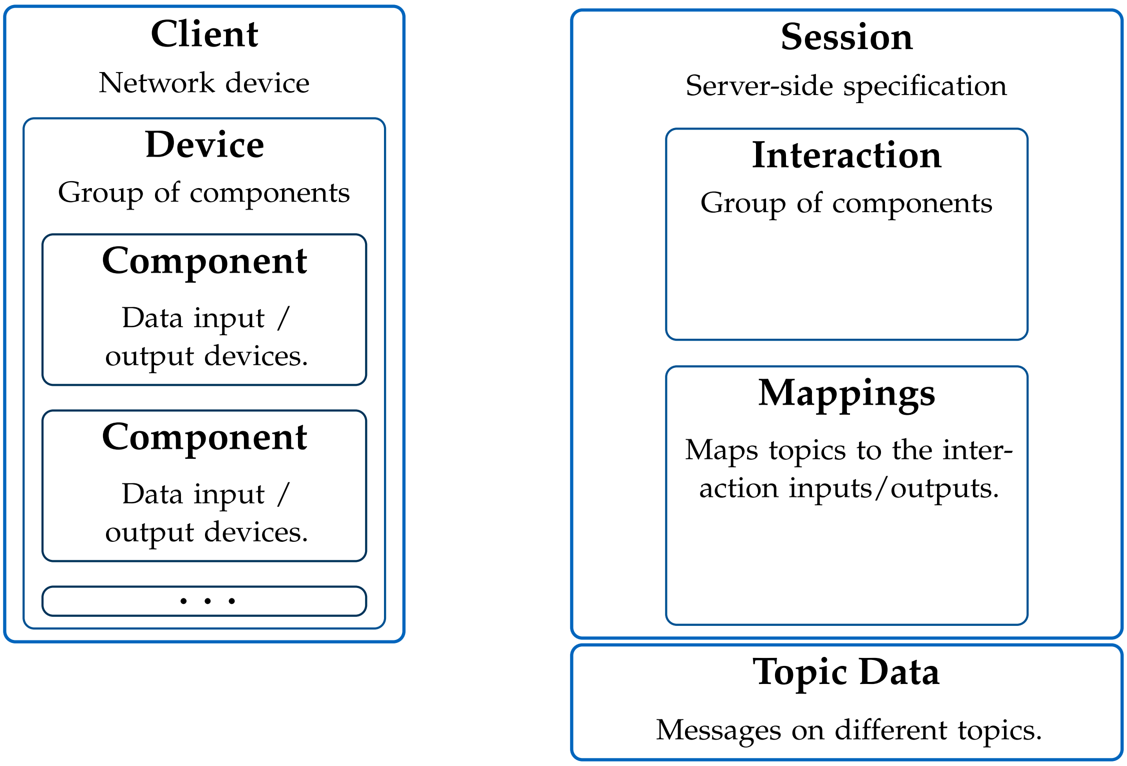


Figure 3.1: Do not forget!

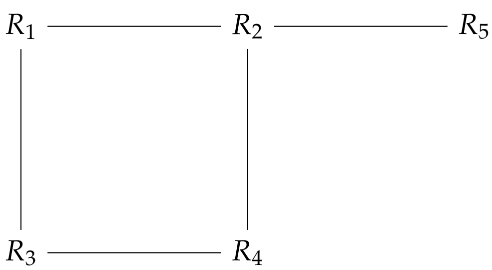


Figure 3.2: An example for a simple drawing.

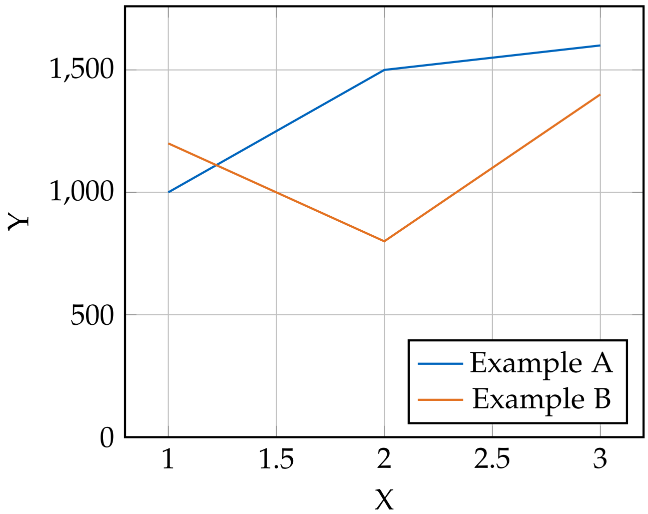


Figure 3.3: An example for a simple plot.

```sql
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 3.4: An example for a source code listing.

List of Figures

List of Tables

Bibliography

[Afo+17] L. Afonso, P. Dias, C. Ferreira, and B. S. Santos. "Effect of hand-avatar in a selection task using a tablet as input device in an immersive virtual environment." In: *2017 IEEE Symposium on 3D User Interfaces (3DUI)*. Piscataway, NJ: IEEE, 2017, pp. 247–248. ISBN: 978-1-5090-6716-9. DOI: 10.1109/3DUI.2017.7893364.

[Dev19] Devices and Sensors Working Group. *DeviceOrientation Event Specification: Editor's Draft, 15 April 2019*. Ed. by Devices and Sensors Working Group. 2019. URL: https://w3c.github.io/deviceorientation/#dom-deviceorientationevent-alpha (visited on 06/17/2019).

[Eva19] Evan You. *Vue.js*. 2019. URL: https://vuejs.org/ (visited on 06/17/2019).

[Goo19] Google LLC. *Protocol Buffers | Google Developers*. 2019. URL: https://developers.google.com/protocol-buffers/ (visited on 06/19/2019).

[Ric19] Ricardo Cabello. *Three.js: JavaScript 3D library*. 2019. URL: https://github.com/mrdoob/three.js/ (visited on 06/17/2019).