Technische Realisierung einer Computer-Simulation für die Reflexion von Lichtstrahlen in einem zweidimensionalem Raum

Ben Weber

November 2021

Inhaltsverzeichnis

1 Einleitung

Die physikalischen Grundlagen für die Reflexion von Lichtstrahlen ist für das menschliche Gehirn einfach zu verstehen. So kann der Mensch die Bahn von reflektierten Lichtwellen an einfachen geometrischen Formen fast intuitiv vorhersagen. In einem komplexeren Umfeld mit einer erhöhten Anzahl an physikalisch relevanten Entitäten wird die Prognose der Gesamtheit aller stattfindenden Einwirkungen auf den Lichtweg jedoch für Menschen zunehmend schwierig zu prognostizieren und vor allem zeitaufwendig. Wenn auch der initiale Aufwand für die digitale Implementierung dieser physikalischen optischen Grundlagen der Reflexion größer sein mag, ist dies vor allem in erwähnten umfangreicheren Situationen praktikabel und ermöglicht schnelle Iterationen bezüglich der Veränderung des Resultats im Zusammenhang mit der Ausgangssituation.

Die vorliegende Arbeit beschäftigt sich mit der Frage, wie eine solche digitale Simulation umgesetzt werden kann. Hierbei wird nicht nur auf die Umsetzung der tatsächlichen Reflexion eingegangen, sondern auch auf den Aufbau einer solchen Computer-Simulation, sodass eine Erweiterung hinsichtlich weiterer strahlenoptischer Phänomene möglich ist.
Zur Vereinfachung wird von einer idealisierten, zweidimensionalen, evakuierten Umgebung ausgegangen, in der Lichtstrahlen nicht an Intensität verlieren und die Oberflächen von Hindernissen störungsfrei sind.

Die Quellenlage bezüglich der hier verwendeten Grundlagen der geome-

trischen Optik ist als sehr sicher zu bewerten. Der Aufbau und die technische Umsetzung der Simulation entstammen fast ausschließlich aus eigenen Überlegungen.

2 Aufbau

Im Folgenden wird der Aufbau der Simulation zunächst theoretisch und abstrahiert betrachtet. Folgende Notation soll jedoch eine Verknüpfung mit dem Quellcode vereinfachen: (`Name im Quellcode`)

2.1 Welt

Die Welt (`World`) ist der Grundstein für die Simulation. Zu diesem virtuellen zweidimensionalen Raum können sämtliche Entitäten hinzugefügt werden. Hier werden diese verwaltet, verarbeitet und zur Visualisierung ausgegeben.

2.2 Lichtstrahl

In der geometrischen Optik geht man bei einem Lichtstrahl (`Ray`) von einem Strahl aus, der sich von einem Ursprung A geradlinig in eine Richtung mit dem Winkel α ausbreitet. (vgl. Tipler 2015, S. 1041) mathematisch wird dieser als Halbgerade angesehen $\overrightarrow{AB}$. In der Simulation ist es sinnvoll lediglich den Startpunkt als Ortsvektor $\vec{O}$ (`origin`) innerhalb der Welt und den Winkel (`angle`), in den der Strahl ausgeht direkt zu speichern.

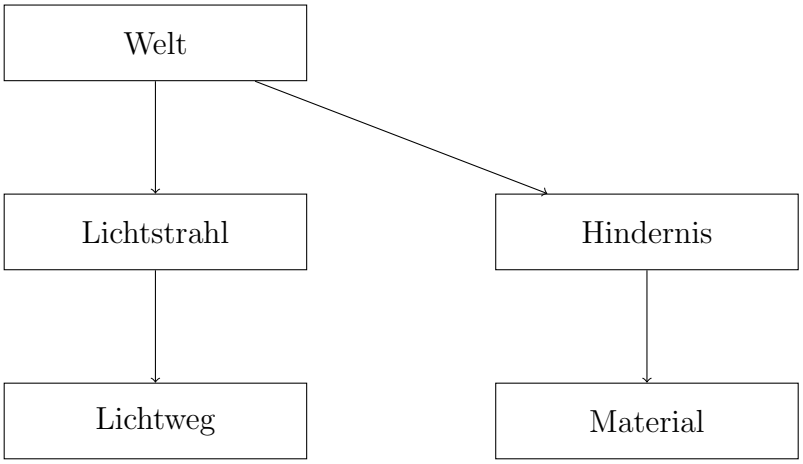

Abbildung 2.1: Schematischer Aufbau der Simulation
Quelle: Eigene Darstellung

2.3 Lichtweg

Es muss davon ausgegangen werden, dass ein Lichtstrahl in der die Kollision mit anderen Objekten seine Richtung verändert. Ab einer Richtungsänderung wird der bisherige Lichtstrahl als Lichtweg (`Line`) gespeichert, um in später zu visualisieren. Hierbei besteht ein Lichtstrahl aus den Ortsvektoren $\vec{O}$ (`origin`) dem Ursprung des Lichtstrahls und $\vec{E}$ (`end`) dem Punkt der Kollision.

2.4 Hindernis

Bei allen Entitäten in der Welt, die nicht Strahl oder Lichtweg sind, handelt es sich um Hindernisse. Dieser definieren sich durch ihre geometrische Form und Position in der Welt und können von Lichtstrahlen getroffen werden. Es sind verschiedene Variationen möglich. Jedes Hindernis (`Obstacle`) hat jedoch einen Startpunkt als Ortsvektor $\vec{A}$ (`start`) in der Welt. Und ein Material (`material`) (vgl. 2.5). Folgende Typen wurden für diese Arbeit ausgewählt,

eine Erweiterung ist jedoch möglich:

2.4.1 Kreis

Der Kreis (`type: 'cirlce'`) hat eine zusätzliche Angabe über den Radius r (`radius`).

2.4.2 Linie

Neben einen Anfangspunkt hat die Linie (`type: 'line'`) auch einen Ortsvektor $\vec{B}$ als Endpunkt.

2.4.3 Kurve

Zusätzlich kann ein Hindernis auch eine Kurve (`type: 'curve'`) sein. Das ermöglicht die Konstruktion verschiedener besonderer Reflektoren (z. B. Parabolischer Reflektor). Für den Verlauf der Kurve besitzt dieser Hindernis-Typ eine Funktion f. Um eine Kurve beliebig in der Welt zu positionieren, kann ihr zusätzlich eine Winkel γ (`rotation`) übergeben werden, der alle Punkte auf Kurve relative zum Ursprung rotiert.
Im Umfang dieser Arbeit wird bewusst auf eine vollständige und genaue Darstellung eines Graphen verzichtet. Daher werden nur Punkte in einem Interval $[-40, 40]$ im Abstand von einer übergebene Skalierung k (`scale`) errechnet, als Linien verbunden und als diese Fortlaufend verarbeitet. Diese Annäherung ist ausreichend, während gleichzeitig der Umfang stark reduziert wird.

2.5 Material

Die Existenz eines Materials (`Material`) in der Welt besteht nur in der Verknüpfung zu einem Hindernis. Ein Material ist für die Verarbeitung eines einfallenden, mit dem Hindernis kollidierenden, Lichtstrahls verantwortlich und kann eine beliebig Anzahl an neuen Lichtstrahlen zurückgeben.
In dieser Arbeit wird nur die Reflexion bei dem Material Spiegel (`mirror`) behandelt. Eine Trennung von Material und Hindernis ermöglicht jedoch eine einfache Erweiterung. Dabei können trotz Veränderung der strahlenoptischer Funktion, die geometrischen Eigenschaften beibehalten werden. So könnte beispielsweise ein Kreis einerseits als Spiegel die Lichtstrahlen reflektieren und anderseits als Glas die Lichtstrahlen brechen. Ohne das dafür die Veränderung des Hindernisses von Nöten wäre.

3 Technische Voraussetzungen

Um die Zugänglichkeit der Simulation zu erhöhen, wird diese als Web-App programmiert. Dazu wird die Programmiersprache TypeScript benutzt und die Applikation wird mit dem Framework „SvelteKit“[1] kompiliert. Zur Visualisierung wird die Library „Pts“[2] verwendet. Diese ermöglicht eine einfache Verwendung der Canvas-API. Zusätzlich werden aber auch einige mathematisch Hilfsfunktionen von dieser Library verwendet.

Der gesamte Quellcode für die Simulation ist auf GitHub aufzufinden[3]. Im Folgenden werden nur kleine, unvollständige Ausschnitte aus dem Code eingebunden. Die fertige Web-App ist ebenfalls als Website aufrufbar[4].

[1] https://kit.svelte.dev
[2] https://ptsjs.org
[3] https://github.com/b3ngg/ray-optics-simulation
[4] https://optics.b3n.gg

4 Simulationsprozess

Im Folgenden wird nun der Ablauf der Simulation vom erstellen der Szene bis hin zum Visualisieren des Resultats beschrieben.

4.1 Erzeugen der Welt

Zunächst muss für die Simulation eine Umgebung errichtet werden. Dazu wird eine Welt erstellt. Zu dieser werden in dieser Szene ein von links oben (man beachte, dass das Koordinatensystem den Ursprung links oben hat und die y-Achse umgekehrt zum normalen Koordinatensystem ist) beginnender und diagonal nach rechts unten strahlender Lichtstrahl hinzugefügt.
Zudem wird auch ein Hindernis in der Form einer vertikalen Linie in der Mitte der Szene hinzugefügt. Diese hat das Material Spiegel (`mirror`).
Nun wird mit `world.update()` die Simulation aktualisiert und die Lichtwege berechnet. Dieser werden zum Schluss visualisiert.

Bei den Objekt `space` handelt es sich um ein von Pts bereitgestellte und zur Visualisierung benötigte Objekt. Die Klasse `Pt` ist ebenfalls Teil von Pts und wird zur Vektormathematik verwendet.

```
/**
 * Simple reflection of a ray on a vertical line
 */
export const lineReflection: Scene = (space) => {
    const form = space.getForm();
    const world = createWorld();

    world.addSource('ray', createRay(new Pt(), Math.PI * 2.25));
    world.addObstacle(
        'line',
        createLine(new Pt(800, 0), { end: new Pt(800, 5000), material: mirror })
    );

    world.update();
    space.add(() => {
        world.draw(form);
    });

    space.playOnce();
};
```

4.2 Berechnung des Lichtwegs

Bei der Berechnung eines Lichtwegs startet die Simulation nur mit einem Ausgangs-Lichtstrahl. Zunächst muss die erste Überschneidung des Strahls mit einem Hindernis bestimmt werden. Für die Bestimmung des Schnittpunktes werden die üblichen Verfahren der Euklidischen Geometrie benutzt. Die Funktionen dafür werden von Pts bereitgestellt und sollen hier nicht weiter vertieft werden.
Der Weg von $\vec{O}$ des Lichtstrahls bis zum Schnittpunkt $\vec{E}$ wird dann als Lichtweg gespeichert und später visualisiert. Aus der Kollision mit dem Hindernis und dem entsprechenden Material können dann weitere Lichtstrahlen resultieren. Diese Lichtstrahlen durchlaufen den gleichen Prozess wie der erste Lichtstrahl. Daher wird die Funktion als rekursive Funktion verwendet und mit den

jeweils neuen Lichtstrahlen erneut aufgerufen.

```
const calculateTraceLines = (
    currentRay: Ray,
    lines: PtIterable[],
    depth: number
): PtIterable[] => {
    // Prevent infinite reflections
    if (depth >= MAX_TRACE_DEPTH) return lines;

    // Get all collisions
    const allCollisions = getAllCollisions(currentRay, obstacles);

    // Get best collision
    const collision = getFirstCollision(allCollisions, currentRay.origin);

    // Return the lines and the current ray if not collision is found
    if (!collision) return [...lines, rayToPts(currentRay)];

    const { intersection, collider, obstacle } = collision;

    // Get new rays resulting of the collision
    const newRays = obstacle.material.handleCollision(intersection, collider, currentRay, obstacle);

    // Trace new rays
    return newRays
        .map((ray) => {
            return calculateTraceLines(ray, [...lines, [currentRay.origin, intersection]], depth + 1);
        })
        .flat();
    };
```

4.3 Berechnung der Reflexion

Bei dem Material des Spiegels wird für jeden eintreffenden Lichtstrahl ein neuer Lichtstrahl mit verändertem Winkel zurückgegeben. Hierbei gilt das Fermat'sche-Prinzip (vgl. Hering 2017, S. 20).

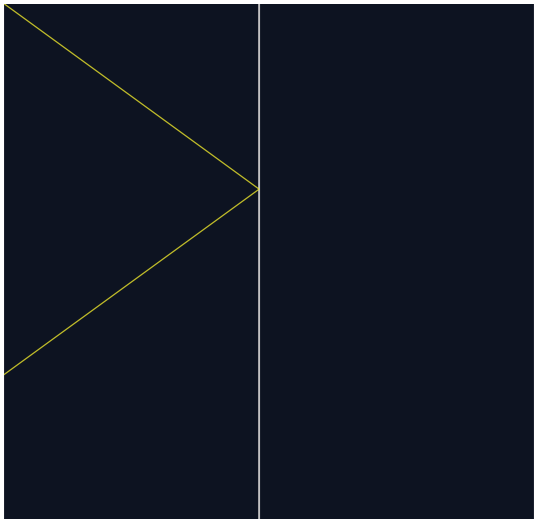

Abbildung 4.1: Reflexion eines Lichtstrahls an einer Linie in der Simulation
Quelle: Eigene Darstellung

An einer Linie

Das Hindernis der Linie kann als eine ebene Grenzfläche angenommen werden. Demnach ist der Betrag des Einfallswinkel ε relative zum Lot der Grenzfläche gleich mit dem Reflexionswinkel ε_r.

$$-\varepsilon = \varepsilon_r \tag{4.1}$$

Es gilt daher zunächst den Lot der Linie zu berechnen. Das Lot ist orthogonal zur Linie. Es kann berechnet werden, indem der Vektor der Linie in zwei Teile zerlegt wird. (vgl. Greve 2006, S. 1)

```
/** Calculate the normal point of a line */
const getLineNormal = (line: PtIterable): Readonly<Pt> => {
    const normalAngle = line[0].$subtract(line[1]).angle() + Math.PI / 2;
    return new Pt(0, 1).toAngle(normalAngle).$unit();
};
```

Mit dem Lot $\vec{N}$ und dem Einfallsvektor $\vec{E}$ lässt sich nun der Reflexionsvektor $\vec{R}$ berechnen. (vgl. Cross 2013, Kapitel 10)

$$\vec{R} = \vec{E} - (\vec{E} \cdot \vec{N})\vec{N} \tag{4.2}$$

Gleichung 4.2 implementiert, sieht wie folgt aus und gibt den neuen reflektierten Lichtstrahl zurück. Die Klasse `Vec` ist von Pts bereitgestellt und beinhaltet Operationen der Vektormathematik.

```
/** Calculate the reflected ray of an incoming ray with the angle d to a line on a point */
const getReflectedRayOnLine = (
    incidentRay: Ray,
    line: PtIterable,
    collisionPoint: Pt
): Readonly<Ray> => {
    const d = new Pt(1, 0).toAngle(incidentRay.angle);
    const lineNormal = getLineNormal(line);

    const perpendicular = 2 * Vec.dot(d, lineNormal);
    const reflection = Vec.subtract(d, Vec.multiply(lineNormal, perpendicular));
    const r = new Pt(reflection);

    return createRay(collisionPoint, r.angle());
};
```

An einem Kreis

Ein Kreis kann als Konvexspiegel mit dem Mittelpunkt M und dem Brennpunkt F angesehen werden. (vgl. Kuchling 2004, S. 362) Wobei bei einem Kreis folgendes gilt:

$$M = F \tag{4.3}$$

4.3 nach ergibt sich für den Reflexionsvektor $\vec{R}$ mit dem Mittelpunkt $\vec{M}$ und dem Schnittpunkt $\vec{E}$ folgende Gleichung:

$$\vec{R} = \vec{E} - \vec{M} \tag{4.4}$$

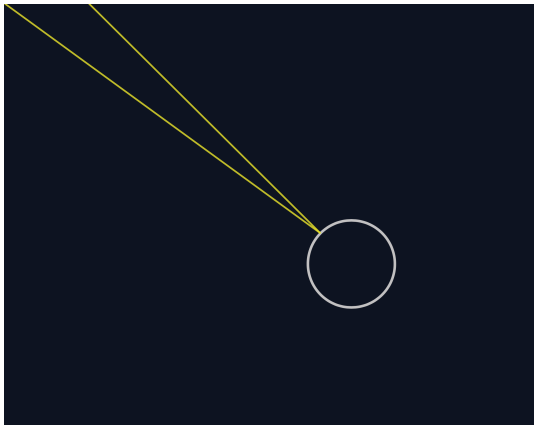

Abbildung 4.2: Reflexion eines Lichtstrahls an einem Kreis in der Simulation

Quelle: Eigene Darstellung

Um Gleichung 4.4 zu implementiert genügt lediglich folgender Quellcode:

```
/** Calculates the reflected ray of an incoming ray on a circle */
const getReflectedRayOnCircle = (collisionPoint: Pt, circleCenter: Pt): Readonly<Ray> => {
    const perpendicularAngle = collisionPoint.$subtract(circleCenter).angle();
    return createRay(collisionPoint, perpendicularAngle);
};
```

Zusatz: Komplexe Umgebungen

Im Folgenden sollen komplexere Umgebungen und deren Resultat in der Simulation demonstriert werden. Weitere interaktive Beispiele sind online[1] aufzufinden.

[1] https://optics.b3n.gg

Parabolischer Reflektor

Bei einer parabolischen Reflektor handelt es sich um einen Konkavspiegel. (vgl. Kuchling 2004, S. 362). Senkrecht einfallende Strahlen werden hierbei zu einem Brennpunkt reflektiert.

```
/**
* Reflections of multiple rays on a parabolic reflector
*/
export const parabolic: Scene = (space) => {
    const form = space.getForm();
    const world = createWorld();

    for (let i = 0; i < 10; i++) {
        world.addSource('ray' + i, createRay(new Pt(500 + 40 * i, 0), Math.PI * 2.5));
    }

    world.addObstacle(
        'reflector',
        createCurve(new Pt(480 + 5 * 40, 500), { f: (x) => (x / 10) ** 2, scale: 6, material: mirror })
    );

    world.update();
    space.add(() => {
        world.draw(form);
    });

    space.playOnce();
};
```

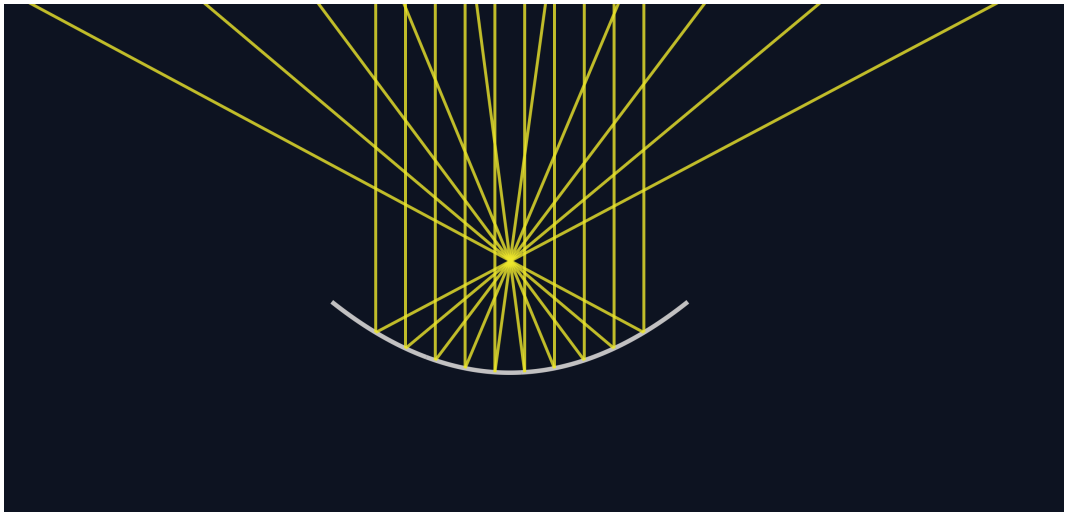

Abbildung 4.3: Reflexion mehrere Lichtstrahlen an einem parabolischen Reflektor

Quelle: Eigene Darstellung

Reflexion an Kreisen

Bei einer großen Anzahl an Kreisen entwickeln sich sehr schnell viele Reflexionen. Eine Art Stress-Test für die Simulation.

```
export const circleChaos: Scene = (space) => {
    const form = space.getForm();
    const world = createWorld();
    space.add({
        start: (bound) => {
            const pts = Create.distributeRandom(bound, 30);
            pts.forEach((pt, i) => {
                world.addObstacle('circle' + i, createCircle(pt, { material: mirror, radius: 60 }));
            });

            world.addSource('dynamic-ray', createRay(space.center, Const.pi));
        }
    });
    space.playOnce();
};
```

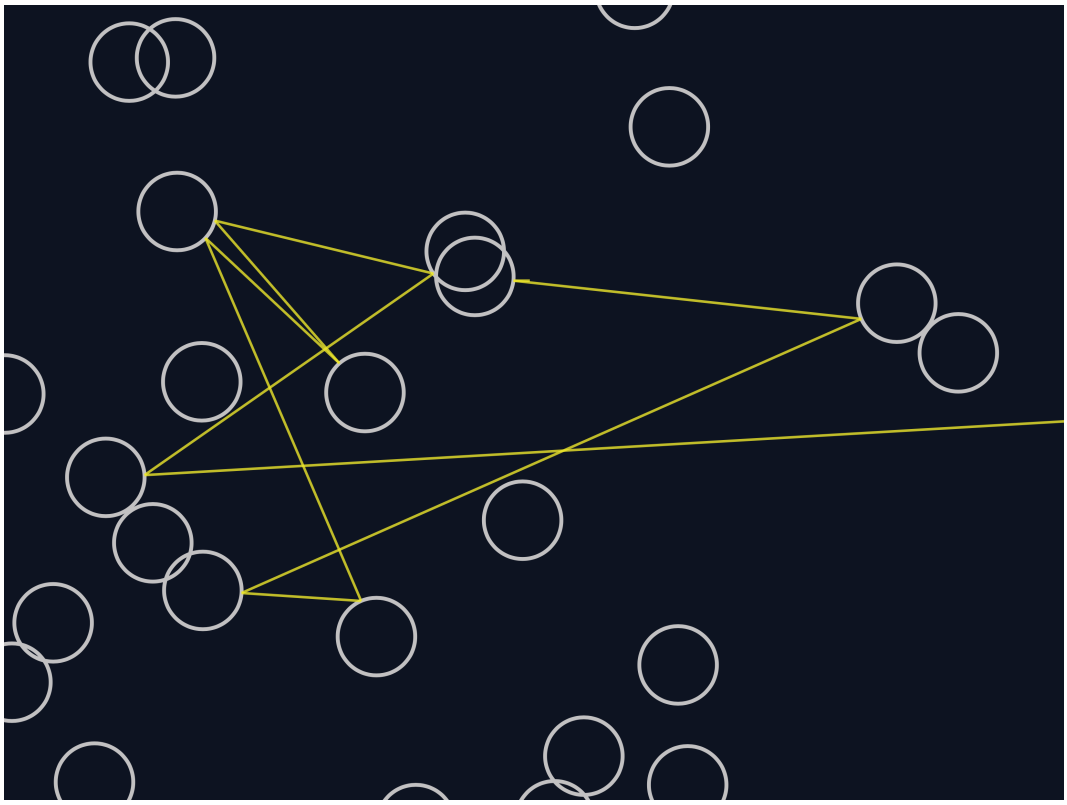

Abbildung 4.4: Reflexion eines Lichtstrahls an vielen Kreisen

Quelle: Eigene Darstellung

5 Fazit

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum

Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache

gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Literatur

Cross, Don (2013). *Fundamentals of Ray Tracing*. http://cosinekitty.com/raytrace/chapter10_reflection.html. [Online; Zugriff: 21.11.21].

Greve, Bram de (2006). *Reflections and Refractions in Ray Tracing*. https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf. [Online; Zugriff: 21.11.21].

Hering, Ekbert (2017). *Optik für Ingenieure und Naturwissenschaftler Grundlagen und Anwendungen*. München: Fachbuchverlag Leipzig im Carl Hanser Verlag. ISBN: 978-3-446-44281-8.

Kuchling, Horst (2004). *Taschenbuch der Physik mit zahlreichen Tabellen*. MünchenWien: Fachbuchverl. Leipzig im Carl-Hanser-Verl. ISBN: 3-446-22883-7.

Tipler, Paul (2015). *Physik für Wissenschaftler und Ingenieure [der Begleiter bis zum Bachelor*. BerlinHeidelberg: Springer Spektrum. ISBN: 978-3-642-54165-0.