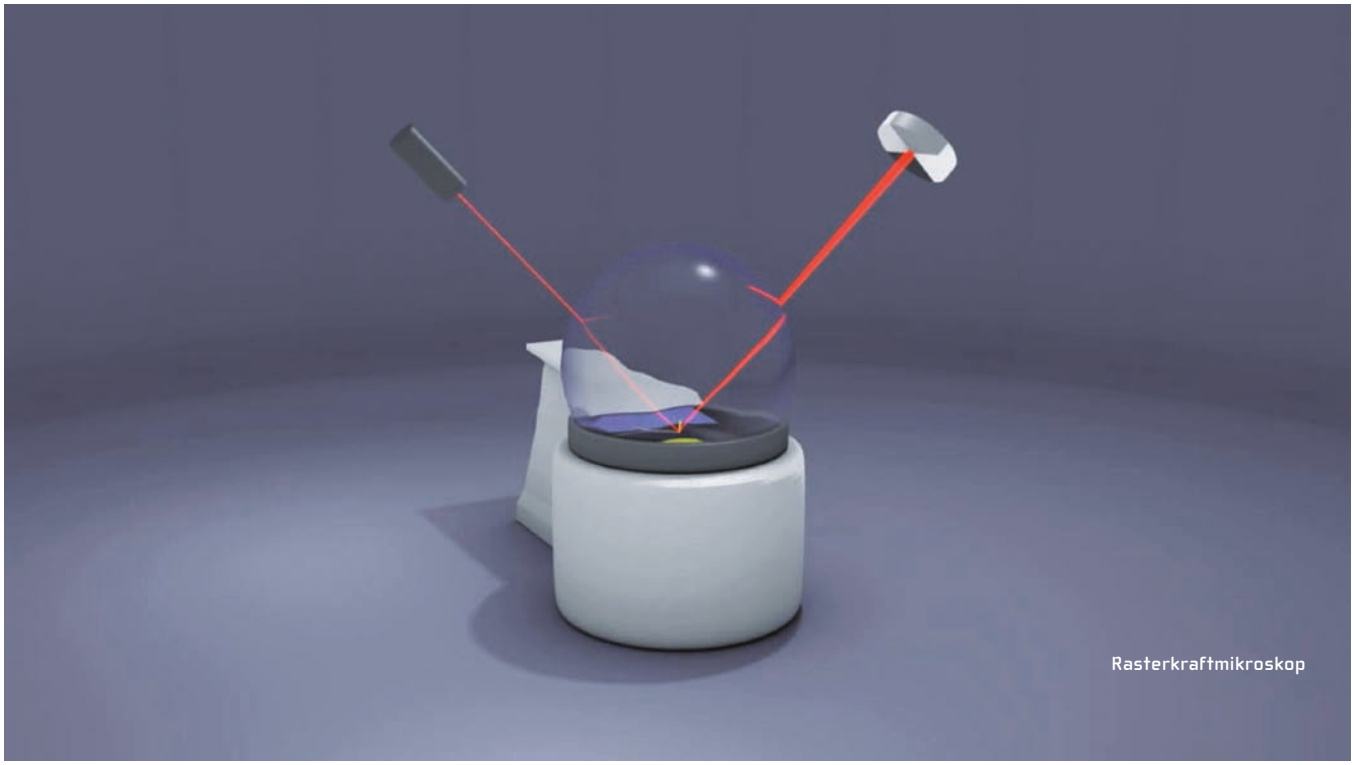


Scripted Laser Reflection

Wie kann Laserlicht in einer computergenerierten Animation realitätsnah simuliert und gesteuert werden? Diese Arbeit behandelt die Simulation und physikalisch korrekte Reflexion eines animierten Laserstrahls an einer beliebigen Ebene.

von Wolfgang Höhl

Der Laserstrahl wurde in zwei Teilen als scriptgesteuertes Extrusionsobjekt hergestellt. Es wird gezeigt, dass speziell Laserlicht durch scriptgesteuerte Extrusionsobjekte wesentlich einfacher und schneller animiert und simuliert werden kann, als mit herkömmlichen Lichtquellen und Renderverfahren. Ziel dieser Arbeit ist die Entwicklung eines scriptbasierten Setups zur realitätsnahen Simulation eines Laserstrahls mit der freien Software Blender.

Die Prinzipien und Ergebnisse sind auf andere Softwarepakete übertragbar und anwendbar. Script-Constraints konnten erfolgreich auf zwei voneinander abhängige Extrusionskörper angewendet werden. Mit diesem Setup ließen sich sehr realitätsnahe Ergebnisse im Verhalten von Laserstrahlen erzielen. Das scriptgesteuerte Setup erwies sich als sehr stabil, flexibel und adaptierbar im Rahmen der gestellten Anforderungen.

Ein Animationsfilm für die Nanowissenschaft

Diese Arbeit entstand im Rahmen meiner Lehrveranstaltung „Praktikum 3D-Modellierung mit Blender" im Sommersemester 2011 am Institut für Informatik der LMU München. Die Aufgabe der Studierenden war die Konzeption, Gestaltung und Umsetzung eines computergenerierten Animationsfilmes für drei aktuelle Doktorarbeiten der Nanosystems Initiative Munich und am Physics Department der Technischen Universität München (TUM). Für die Doktorarbeiten von Bizan N. Balzer und Sandra Kienle sollten mindestens zwei 3D-Visualisierungen umgesetzt werden: „Aufbau eines Rasterkraftmikroskops und Animation von Haftungsmessungen eines Einzelmoleküls mit einem Rasterkraftmikroskop" und „Animation von Reibungsmessungen mit einem Rasterkraftmikroskop". Ein Team von Studierenden setzten beide Themen im computergenerierten Animationsfilm „Atomic Force Microscopy based Single Molecule Desorption" um.

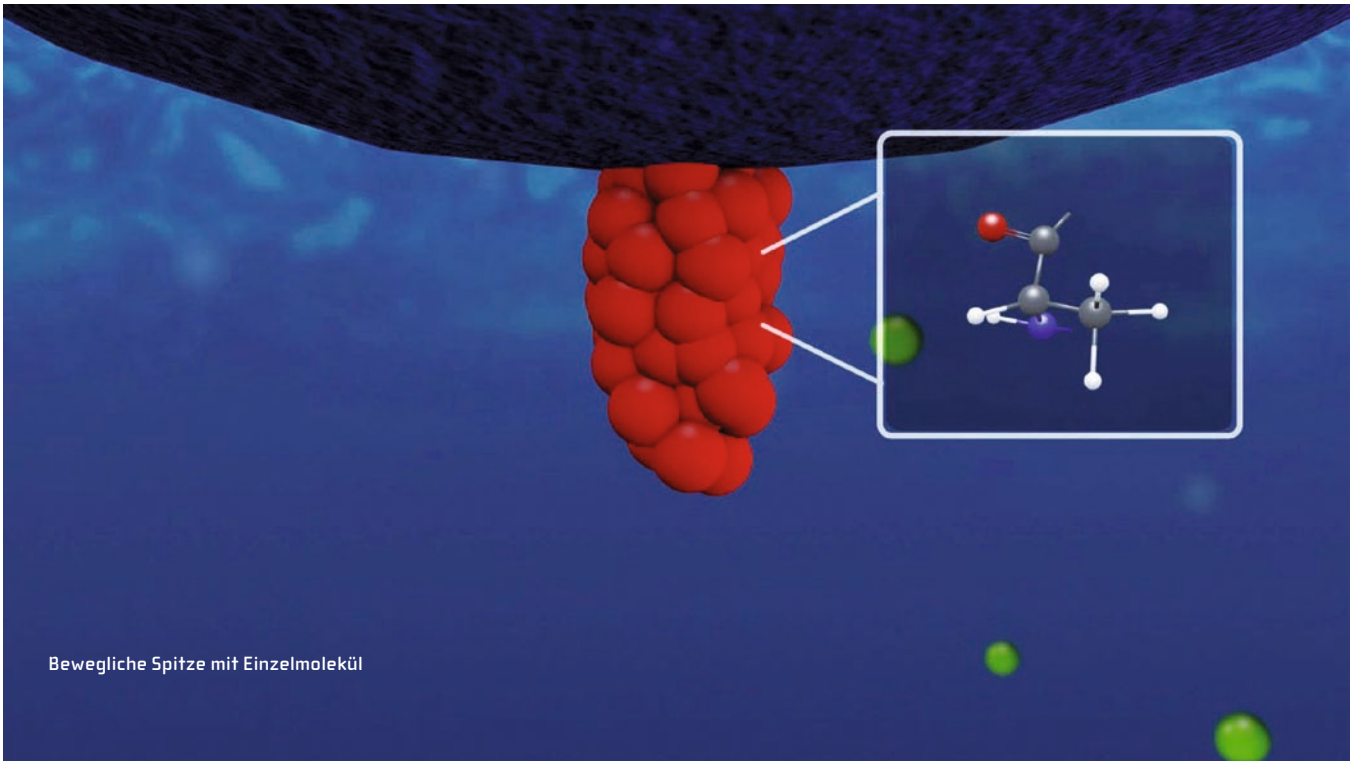
Laser Reflection – Setup und Anforderungen

Im vorgenannten Animationsfilm soll ein Laserstrahl simuliert werden, der an der Rückseite der beweglichen Spitze des Rasterkraftmikroskops reflektiert wird und anschließend auf eine Viersegment-Photodiode trifft. Der schematische Aufbau des Rasterkraftmikroskops ist in nachfolgender Abbildung dargestellt. Hierbei ist darauf zu achten, dass in jedem Abschnitt der Animation das Reflexionsgesetz eingehalten wird, also Ein- und Ausfallswinkel des Laserstrahls stets gleich groß sind.

Des Weiteren muss die Länge des Laserstrahls an die sich ändernde Entfernung der Spitze von der Laserdiode angepasst werden. Im Folgenden werden einige verwandte Arbeiten genannt und die Vorüberlegungen sowie die Implementierung des Lasers unter Verwendung von Script-Constraints beschrieben.

Best Practice – scriptgesteuerte Lichtsimulation

Für die scriptgesteuerte Tageslichtsimulation bieten viele Softwarepakete bereits umfangreiche Werkzeuge an. Auch für die Darstellung von Laserstrahlen gibt es bereits Lösungen. Allerdings befinden sich darunter nur wenige benutzerfreundliche Ansätze für die physikalisch korrekte und realitätsnahe Simulation der Reflexion von Laserstrahlen für computergenerierte Animationsfilme.



Bewegliche Spitze mit Einzelmolekül

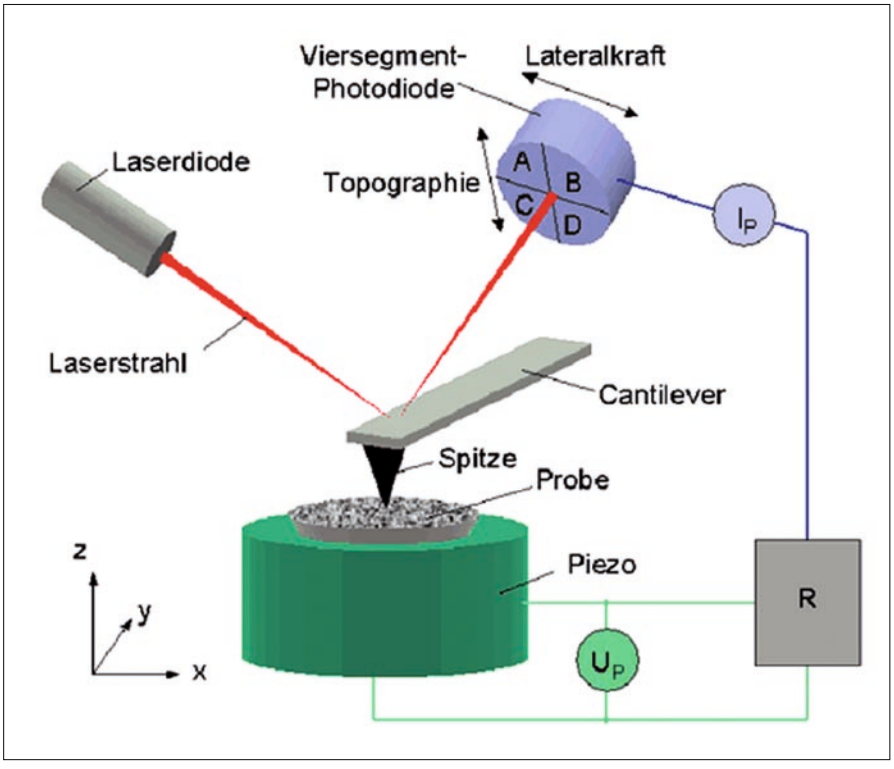
Im Laser-Beam-Tutorial des New Blender Materials Archive wird der Ansatz gewählt, den Strahl als Objekt zu erstellen und dieses durch ein geeignetes Material wie einen Laser aussehen zu lassen. Reflexionen werden hier nicht behandelt. In 3ds Max lassen sich Laserstrahlen mit Hilfe von Zielrichtungslichtern und dem Volumenlicht-Effekt konstruieren. Diese stellen automatisch realistische Reflexion und Brechung der Strahlen zur Verfügung. Immel und Huff zeigen, dass sich auch mit dem Raytracer POV-Ray Laserstrahlen mit Hilfe von Photonen generieren lassen. Damit lassen sich Reflexion und Brechung realistisch simulieren.

Laserstrahl als script-gesteuertes Extrusionsobjekt

Um einen parallelen Strahl des Lasers zu simulieren, wird eine Kreisform entlang eines geraden Pfads extrudiert, um ein zylinderförmiges Extrusionsobjekt zu erhalten. Die Simulation des Lasers durch eine Lichtquelle erschien ungeeignet, da sich in Blender nur mit Hilfe des Sonnenlichts („Sun Lamp") paralleles Licht simulieren lässt. Um einen gebündelten Strahl zu erhalten, müsste dieses durch geeignete Geometrie eingeschränkt werden. Zusätzlich müsste, um den Strahl sichtbar zu machen, ein so genannter „Halo-Effekt" (Sichtbares Licht) angewandt werden. Dieser lässt sich jedoch nicht reflektieren, wodurch der Reflexionsstrahl nicht sichtbar gemacht werden kann.

Deshalb werden der Strahl und dessen Reflexion mit Hilfe zweier Extrusionsobjekte simuliert. Hierfür wird den beiden Objekten ein rotes, leicht transparentes, emittierendes Material zugewiesen. Aufgrund der Verwendung von Objekten zur Simulation des Laserstrahls muss deren Verhalten durch Constraints festgelegt werden.

Beispielsweise muss der Laserstrahl immer direkt auf der Oberfläche des Cantilevers enden. Der Reflektionsstrahl hingegen beginnt genau in diesem Punkt und endet in der Ebene der Photodiode. Die beiden Strahlen müssen gemäß dem Brechungsgesetz mit der Normalen der Oberfläche jeweils den gleichen Winkel einschließen. Diese Abhängigkeiten werden durch eine Kombination von Standard- und Script-Constraints gelöst. Die nachfolgende Implementierung ist in zwei Teile untergliedert. Zunächst wird der erste Teil des Strahls von der Laserdiode zum Cantilever betrachtet (einfallender Strahl). Anschließend wird die Reflexion des Strahls behandelt (reflektierter Strahl).



Schematischer Aufbau Rasterkraftmikroskop

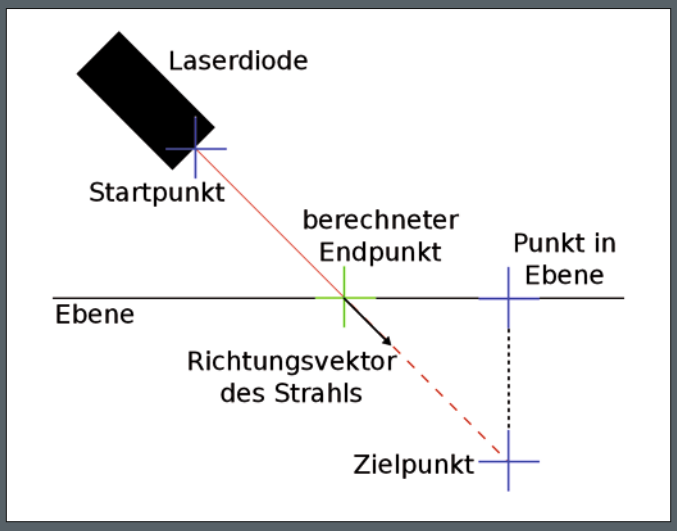
Scripted Laser Reflection – Team

Projekt

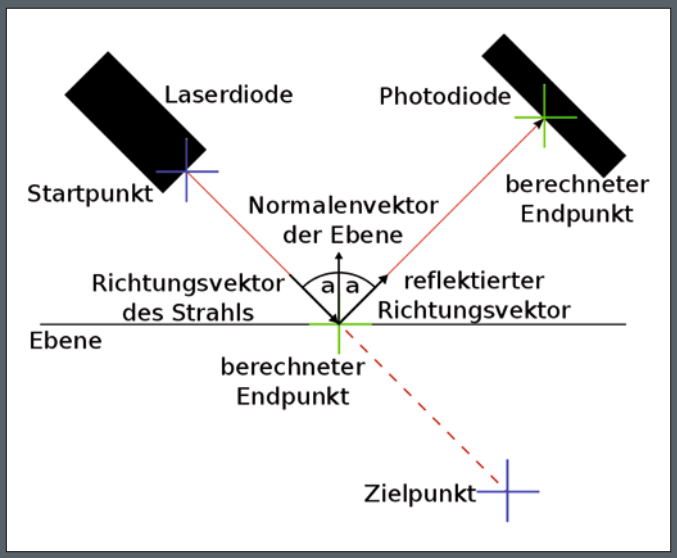
- Simulation der Reflexion eines Laserstrahls an einer beliebigen Ebene mit Script-Constraints: Benedikt Bleimhofer, Wolfgang Höhl (bleimhofer@cip.ifi.lmu.de, hoehl@cip.ifi.lmu.de);
- Ludwig-Maximilians-Universität München (LMU), Institut für Informatik | Lehr- und Forschungseinheit für Medieninformatik, Amalienstraße 17, 80333 München

Teammitglieder

Benedikt Bleimhofer, Philip Czech, Jonas Hartmann, Lukas Höfer, Lukas Kappeler und Sylvia Kempe. Ich bedanke mich bei diesen Studierenden für ihren Einsatz und ihr hervorragendes Engagement beim Umsetzen dieses Animationsfilmes. Diese Forschungsarbeit entstand gemeinsam mit Benedikt Bleimhofer, der sich hauptsächlich mit der Entwicklung und Umsetzung der Lasersimulation beschäftigte. Mein besonderer Dank geht an Herrn Prof. Dr. Thorsten Hugel, Bizan N. Balzer, Sandra Kienle vom E22a, TUM Physics Department (www.bio.ph.tum.de) und an die Nanosystems Initiative Munich, Prof. Jochen Feldmann, Dr. Peter Sonntag und Christoph Hohmann. Ohne die gemeinsame Arbeitsplattform und die Vermittlung der Doktorarbeiten wäre diese Forschungsarbeit nicht möglich gewesen. Insbesondere danke ich auch Univ. Prof. Dr. Heinrich Hußmann und Univ. Prof. Dr. Andreas Butz für die Erteilung des Lehrauftrages und die wertvollen Hinweise, die zum Entstehen dieser Publikation beigetragen haben.



Lage der Empty-Objekte (Einfallender Strahl)



Lage der Empty-Objekte (Reflektierter Strahl)

Einfallender und reflektierter Strahl

Zunächst muss der Pfad des Extrusionsobjekts auf den Bereich zwischen Laserdiode und Cantilever angepasst werden. Dies geschieht, indem Start- und Endpunkt über „Hooks" an „Empty-Objekte" fixiert werden. Der Startpunkt wird dabei an den Austrittspunkt des Lasers an der Diode gesetzt. Der Endpunkt wird an ein Empty-Objekt fixiert, das selbst über ein Script-Constraint kontrolliert wird. Dieses wird im Folgenden kurz erläutert. Das Script erhält als Eingabe-Parameter („Targets") drei weitere Empty-Objekte.

Als erster Parameter wird ein Objekt übergeben, das den Z-Wert, die Höhe der Ebene, in der der Strahl reflektiert werden soll, angibt. Dies wird erreicht, indem das Objekt in der Ebene respektive an deren Oberfläche platziert wird. Als zweiter und dritter Parameter werden der Start- und der Zielpunkt des Lasers übergeben. Diese legen die Ausrichtung des Lasers fest. Der Zielpunkt legt dabei nur die Richtung und nicht den tatsächlichen Endpunkt des Lasers fest.

So lässt sich die Richtung des Lasers unabhängig von der Lage des Schnittpunkts entlang des Strahlverlaufs steuern. Zusätzlich wird der Zielpunkt mit der Rotation der Ebene verknüpft, um im zweiten Schritt die Berechnung des Einfallswinkels zu ermöglichen. Zuerst wird nun mit Hilfe des Start- und des Zielpunkts der Richtungsvektor des Strahls ermittelt. Anschließend wird der Z-Abstand in Vielfachen des Richtungsvektors zwischen dem ersten Parameter (dem Punkt in der Reflexionsebene) und dem Zielpunkt ermittelt. Der tatsächliche Endpunkt des Lasers in der Ebene ergibt sich anschließend durch Addieren des mit dem erhaltenen Faktor skalierten Richtungsvektors zum Zielpunkt.

Die obige Abbildung zeigt ein Beispiel für die Lage der Objekte in einer 2D-Szene. Der

Wolfgang Höhl hat an der TU Wien Architektur studiert und an der Universität Hannover promoviert. Er arbeitete als Architekt bei von Gerkan, Marg + Partner Architekten in Hamburg und war wissenschaftlicher Mitarbeiter an der TU München.
Heute ist er Professor für Computeranimation an der Macromedia Hochschule für Medien und Kommunikation. Er unterrichtet 3D-Visualisierung an der LMU, an der FH Joanneum und an der Hochschule für angewandte Wissenschaften München. Als freischaffender Architekt und Autor schreibt er für Computer- und Architekturfachzeitschriften.

Punkt in der Ebene kann zur Vereinfachung, wie in der Abbildung angedeutet, mit Hilfe eines Location-Constraints auf dem X- und Y-Wert direkt über dem Zielpunkt positioniert werden.

Simulation mit Script-Constraints

Der Reflexionsstrahl ist als eigenes Extrusionsobjekt modelliert. Der Startpunkt des Pfads ist dabei an den (wie im vorigen Abschnitt beschriebenen) berechneten Endpunkt des Strahls in der Ebene fixiert. Für den Reflexionsstrahl muss nun noch die Ausrichtung sowie der Endpunkt auf der Photodiode berechnet werden. Dies geschieht anhand eines weiteren Script-Constraints. Dieses erhält als Parameter den Start-, End- und Zielpunkt des Laserstrahls sowie drei Punkte in der Ebene der Photodiode.

Mit Hilfe des Start- und Endpunktes kann der Richtungsvektor des einfallenden Laserstrahls berechnet werden. Der Zielpunkt ist, wie im vorigen Abschnitt bereits erwähnt, mit der Rotation der Ebene verknüpft. Somit lässt sich der Normalenvektor der Ebene und anhand dessen die Richtung des Reflexionsstrahls ermitteln. Mittels des Richtungsvektors und des Endpunkts kann nun die Geradengleichung für den Reflexionsstrahl aufgestellt werden. Die Ebenengleichung der Photodiode lässt sich unter Verwendung der drei als letzte Parameter übergebenen Punkte aufstellen. Der Schnittpunkt des Reflexionsstrahls mit der Photodiode lässt sich nun durch Gleichsetzen der beiden Gleichungen berechnen.

Durch zwei Script-Constraints sowie einige Standard-Constraints lässt sich so die Reflexion eines Laserstrahls an einer beliebigen Ebene simulieren. In der Darstellung rechts findet sich der Quellcode der Script-Constraints. Es sind jedoch nur die Teile angegeben, um die das von Blender zur Verfügung gestellte Template erweitert wurde.

Flexibles und adaptierbares Rig mit Script-Constraints

Script-Constraints konnten in diesem Setup erfolgreich auf zwei voneinander abhängige Extrusionskörper angewendet werden. Damit ließen sich nicht nur sehr realitätsnahe Ergebnisse im Verhalten und der Darstellung von Laserstrahlen erzielen, das vorliegende Setup erwies sich auch in der Herstellung der computergenerierten Animation als flexibles und schnell adaptierbares Rig.

An ebenen Flächen konnten damit sehr gute Ergebnisse erzielt werden. Als weitere Forschungsperspektive bietet sich die Anwendung desselben Setups auf gekrümmte Oberflächen an. > sha

Source Code der Pythonscripts

```
def doConstraint(obmatrix, targetmatrices, idprop):
  # ...
  # Do stuff here, changing obloc, obrot, and obsca.

  # NUM_TARGETS muss auf 3 gesetzt werden
  # 1. Target: Punkt in Ebene
  # 2. Target: Startpunkt
  # 3. Target: Zielpunkt

  punktInEbene = targetmatrices[0].translationPart()

  startPunkt = targetmatrices[1].translationPart()
  zielPunkt = targetmatrices[2].translationPart()

  # Berechne Richtungsvektor des Strahls
  richtungsVektor = startPunkt - zielPunkt
  richtungsVektor.normalize

  # Berechne Faktor der Abweichung in Z-Richtung
  faktor = (punktInEbene.z - startPunkt.z) / richtungsVektor.z

  # Berechne Endpunkt
  obloc.x = startPunkt.x + faktor*richtungsVektor.x
  obloc.y = startPunkt.y + faktor*richtungsVektor.y
  obloc.z = startPunkt.z + faktor*richtungsVektor.z

  # Convert back into a matrix for loc, scale, rotation,
  # ...
```


Listing 1: Script zum Anpassen des Strahls zwischen Laserdiode und Cantilever

```
def doConstraint(obmatrix, targetmatrices, idprop):
  # ...
  # Do stuff here, changing obloc, obrot, and obsca.

  # NUM_TARGETS muss auf 6 gesetzt werden
  # 1. Target: Startpunkt
  # 2. Target: Endpunkt
  # 3. Target: Zielpunkt
  # 4. Target: 1. Punkt auf Detektor
  # 5. Target: 2. Punkt auf Detektor
  # 6. Target: 3. Punkt auf Detektor

  # Richtungsvektor des einfallenden Strahls berechnen
  startPunkt = targetmatrices[0].translationPart()
  endPunkt = targetmatrices[1].translationPart()
  richtungsVektor = startPunkt - endPunkt
  richtungsVektor.normalize

  # Normale der Reflexionsebene berechnen
  tmpVektor = Mathutils.Vector(0.0,0.0,1.0)
  tmpVektor.normalize
  # 3. Target: Zielpunkt enthaelt Rotation der Ebene
  normalenVektor = tmpVektor * targetmatrices[2].rotationPart()

  # Projektionsmatrix zur orthographischen Projektion des
    Richtungsvektors auf die Reflexionsebene
  projektionsMatrix =
    Mathutils.OrthoProjectionMatrix("r",3,normalenVektor)

  # Berechne Projektion und Reflexion des Richtungsvektors
  projektionsVektor = projektionsMatrix * richtungsVektor
  reflexionsVektor = (richtungsVektor - projektionsVektor) +
    projektionsVektor.negate()
  linevect = reflexionsVektor.copy()
  linevect.negate()

  # Berechne Schnittpunkt mit Photodiode
  # Ebenevektoren der Photodiode:
  ebenenVektor1 = targetmatrices[4].translationPart()
    -targetmatrices[3].translationPart()
  ebenenVektor2 = targetmatrices[5].translationPart()
    -targetmatrices[3].translationPart()

  # Matrizen zur Schnittpunktberechnung:
  matrix0 = Mathutils.Matrix(ebenenVektor1,ebenenVektor2,
    endPunkt-targetmatrices[3].translationPart())
  matrix1 = Mathutils.Matrix(ebenenVektor1,ebenenVektor2,linevect)
  faktor = matrix0.determinant() / matrix1.determinant()

  # Berechne Endpunkt des Reflexionsstrahls auf Photodiode
  obloc.x = endPunkt.x + faktor * reflexionsVektor.x
  obloc.y = endPunkt.y + faktor * reflexionsVektor.y
  obloc.z = endPunkt.z + faktor * reflexionsVektor.z

  # Convert back into a matrix for loc, scale, rotation,
  # ...
```


Listing 2: Script zur Berechnung der Endpunkte des reflektierten Strahls