



Abbildung 1: Durch Raycasting erzeugtes Bild eines komplexeren, texturierten Modells mit Phong-Shading.

CG/task1b—Erweitertes Raycasting

Um realistisch wirkende Bilder wie das in Abbildung 1 zu erzeugen, ist es essentiell, den Effekt von Licht und Schatten auf die Objekte in einer virtuellen Szene zu modellieren. Eine vollständige Simulation sämtlicher physikalischer Aspekte der Interaktion von Licht und Materie ist allerdings in der Regel nicht praktikabel. Um mit begrenzten Ressourcen ein möglichst gutes Ergebnis zu erzielen, bedient man sich physikalisch inspirierter, aber stark vereinfachter Beleuchtungsmodelle um die Farben von Objekten zu bestimmen. Ein einfaches und in der Praxis weit verbreitetes Beleuchtungsmodell ist Phong-Shading.

1 Shading Grundlagen

Der Vorgang der Berechnung der Farbe eines Punktes eines Objektes wird im Allgemeinen als *Shading* bezeichnet. Der Einfachheit halber betrachtet man oft nur den Einfluss der direkten Beleuchtung der Objektoberfläche durch idealisierte, punktförmige Lichtquellen. Wie in Abbildung 2 illustriert, reduziert das Problem sich so auf die Berechnung des in

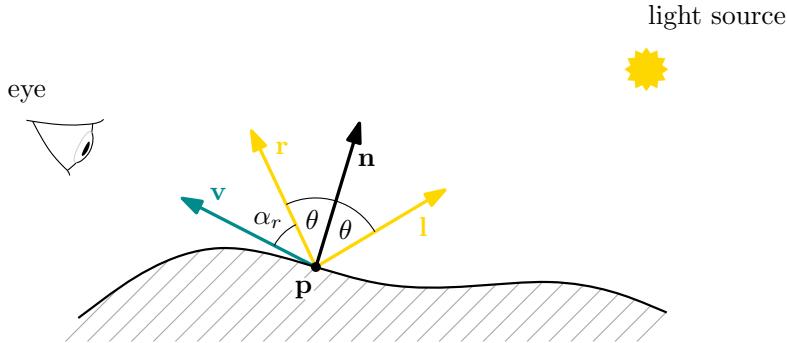


Abbildung 2: Lokale Betrachtung der Beleuchtung eines Punktes **p** auf der Oberfläche eines Objektes durch eine einzelne Lichtquelle. Der in Richtung **v** zum Betrachter reflektierte Anteil des aus Richtung **l** einfallenden Lichts hängt wesentlich vom Einfallswinkel θ ab. Die Normale **n** bestimmt die Ausrichtung der Oberfläche zur Lichtquelle und nimmt daher eine zentrale Stellung in der Beleuchtungsberechnung ein. Der Winkel α_r zwischen der Richtung **v** zum Betrachter und der reflektierten Lichteinfallsrichtung **r** dient als Maß für den spekularen Anteil, während der diffuse Anteil nur von θ abhängt.

Richtung **v** zum Betrachter reflektierten Anteils des aus Richtung **l** von einer Lichtquelle auf den Punkt **p** einfallenden Lichtes. Die Beleuchtung durch mehrere Lichtquellen folgt dem Superpositionsprinzip und entspricht daher einfach der Summe der Effekte jeder einzelnen Lichtquelle.

Das Reflexionsverhalten einer Objektoberfläche wird in der Regel als aus einem diffusen Anteil und einem spekularen Anteil zusammengesetzt modelliert. Der diffuse Anteil c_d entspricht dem Teil des einfallenden Lichtes, der gleichmäßig in alle Richtungen zerstreut wird. Er kann nach dem Lambertschen Gesetz ermittelt werden und hängt nur vom Lichteinfallswinkel θ auf die Oberfläche ab, deren Ausrichtung zur Lichtquelle durch die Oberflächennormale **n** beschrieben ist:

$$c_d = k_d \cdot \max(\cos \theta, 0); \quad (1)$$

k_d ist dabei der diffuse Reflexionskoeffizient des Materials. Der spekulare Anteil entspricht dem Teil des einfallenden Lichtes, der vorwiegend in die um die Oberflächennormale gespiegelte Lichtrichtung **r** reflektiert wird. Im Phong-Shading-Modell wird der in Richtung **v** reflektierte, spekulare Anteil c_s basierend auf dem Winkel α_r zwischen **v** und **r** abgeschätzt:

$$c_s = \begin{cases} k_s \cdot (\max(\cos \alpha_r, 0))^m & \text{wenn } \cos \theta > 0 \\ 0 & \text{sonst} \end{cases} \quad (2)$$

wobei k_s der spekulare Reflexionskoeffizient des Materials ist. Der Faktor m moduliert die Schärfe der spekularen Reflexion und entspricht damit der Glattheit der Oberfläche; je größer m , desto schärfere die spekulare Reflexion. Abbildung 3 zeigt die einzelnen Komponenten der Beleuchtungsberechnung für das Rendering aus Abbildung 1.

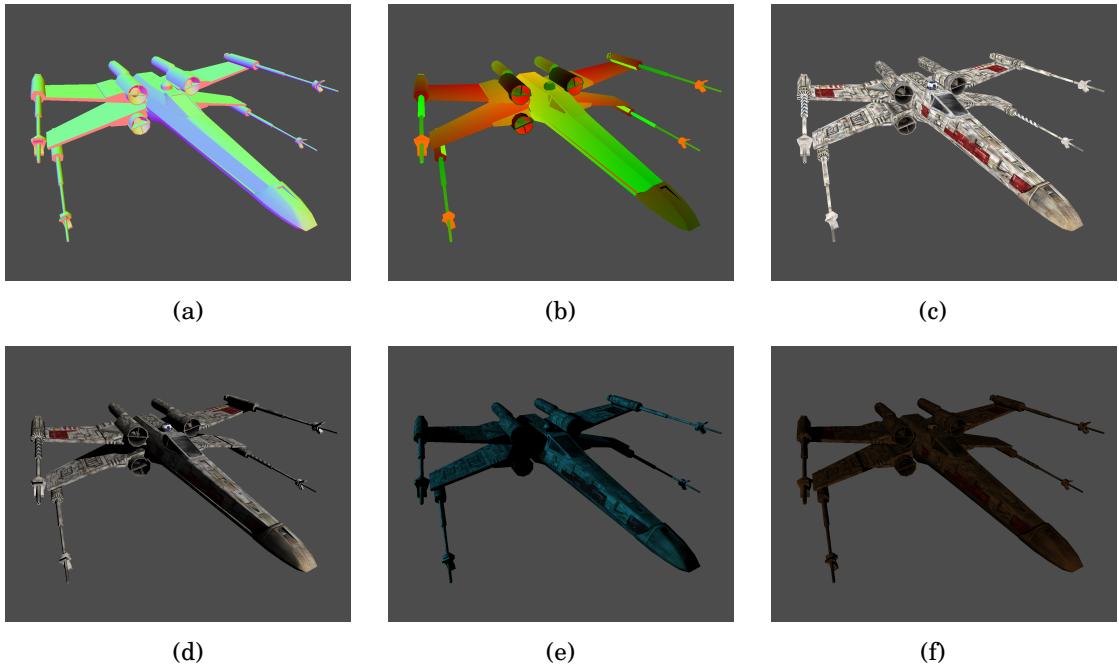


Abbildung 3: Komponenten der Beleuchtungsberechnung. Farbcodierte Visualisierung von (a) Oberflächennormale und (b) Texturkoordinaten sowie (c) der diffuse Reflexionskoeffizient; (d), (e), (f) Beiträge der drei Lichtquellen der Szene

1.1 Interpolation von Oberflächenparametern

Dreiecksnetze bieten einen universellen Ansatz zur Repräsentation von Objektoberflächen. Jede beliebig geformte Oberfläche lässt sich beliebig genau durch ein Dreiecksnetz approximieren, wobei eine bessere Approximation im Allgemeinen mehr Dreiecke und damit mehr Ressourcen benötigt. Zur Berechnung des Shading eines Punktes, ist die Kenntnis verschiedenster Parameter wie beispielsweise der Oberflächennormale oder Texturkoordinaten an diesem Punkt erforderlich. In einer Dreiecksnetzrepräsentation werden solche Parameter in der Regel für jeden Vertex des Netzes gespeichert und die Werte an einem bestimmten Punkt auf einem Dreieck dann durch lineare Interpolation bestimmt.

Abbildung 4 zeigt eine Approximation einer Freiformfläche durch ein Dreiecksnetz. Würde man in den Beleuchtungsberechnungen für ein solches Dreiecksnetz einfach direkt mit den Normalen der Dreiecksflächen arbeiten, würde man ein Ergebnis wie in Abbildung 5a erhalten, man spricht dabei auch von *Flatshading*. Ein besseres Ergebnis lässt sich erzielen, indem man stattdessen an jedem Vertex des Dreiecksnetzes den Normalvektor an die approximierte Oberfläche speichert und in der Beleuchtungsberechnung an jedem Punkt mit einer aus den umliegenden Vertexnormalen interpolierten Normale arbeitet. Wie in Abbildung 5b zu sehen, ist an der Silhouette immer noch erkennbar, dass das Objekt als Dreiecksnetz aufgebaut ist, durch die weiche Schattierung ergibt sich allerdings bereits

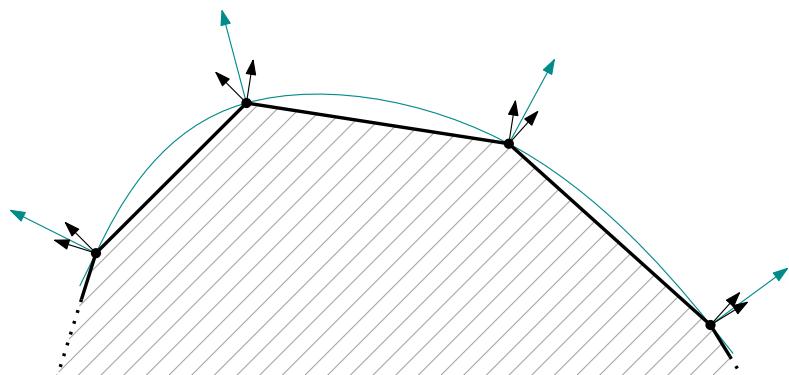


Abbildung 4: Approximation einer Freiformfläche durch ein Polygonnetz (Querschnitt). Zu sehen sind (cyan) die Normalvektoren der angenäherten Oberfläche sowie (schwarz) die Normalvektoren der Polygonflächen an den Vertices des Polygonnetzes.

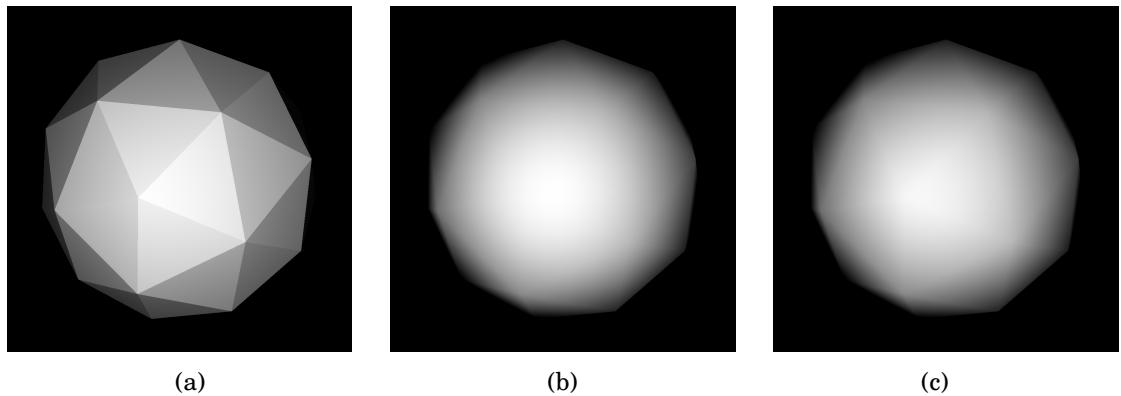


Abbildung 5: Vergleich zwischen (a) Flatshading, (b) Smoothshading und (c) Smoothshading ohne Renormalisierung der interpolierten Normalvektoren einer durch ein Dreiecksnetz angenäherten Kugel.

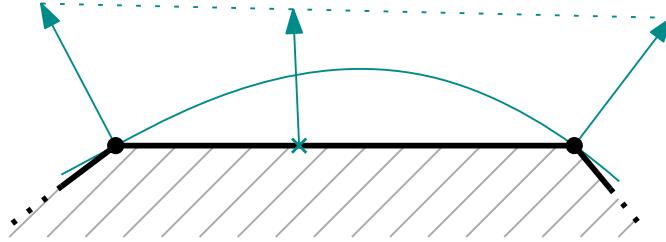


Abbildung 6: Bestimmung der Oberflächennormale zu einem Punkt auf dem Polygongitter durch lineare Interpolation der an den Vertices gegebenen Oberflächennormalen. Man beachte, dass das Ergebnis der linearen Interpolation im Allgemeinen nicht mehr von Einheitslänge ist.

mit relativ wenigen Dreiecken der Eindruck einer runden Oberfläche. Man spricht bei solchen Verfahren, die auf der Interpolation der Beleuchtung oder Beleuchtungsparameter über das Dreiecksnetz basieren, auch von *Smoothshading*.¹

Bei der linearen Interpolation von Oberflächennormalen gilt es, wie in Abbildung 6 dargestellt, zu beachten, dass das Ergebnis der linearen Interpolation zweier Vektoren von Einheitslänge im Allgemeinen kein Vektor von Einheitslänge ist. Bei der Implementierung von Beleuchtungsberechnungen geht man in der Regel davon aus, dass alle Richtungsvektoren inklusive der Oberflächennormalen normalisiert vorliegen, da dies viele Berechnungen vereinfacht. Es ist daher wichtig, das Ergebnis der linearen Interpolation von Oberflächennormalen zu renormalisieren. Ohne eine solche Renormalisierung treten Beleuchtungsartefakte wie in Abbildung 5c zu sehen auf, wo Dreiecksflächen in ihrem Inneren dunkler erscheinen und die Kanten des Dreiecksnetzes hervortreten.

1.1.1 Baryzentrische Koordinaten

Jeder beliebige Punkt \mathbf{p} in der Ebene eines durch die Eckpunkte $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ aufgespannten Dreiecks lässt sich als gewichtete Summe der Eckpunkte ausdrücken:

$$\mathbf{p} = (1 - \lambda_1 - \lambda_2) \cdot \mathbf{p}_1 + \lambda_1 \cdot \mathbf{p}_2 + \lambda_2 \cdot \mathbf{p}_3. \quad (3)$$

Die Gewichte λ_1 und λ_2 bezeichnet man auch als Baryzentrische Koordinaten² des Punktes \mathbf{p} (siehe auch Abbildung 7a). Baryzentrische Koordinaten haben eine Reihe sehr nützlicher Eigenschaften; unter anderem können sie direkt verwendet werden, um zwischen an den Eckpunkten definierten Werten linear zu interpolieren. Seien $\alpha_1, \alpha_2, \alpha_3$ mit den Eckpunkten $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ assoziierte Werte, dann ergibt sich an jedem Punkt \mathbf{p} in der

¹Die konkrete Variante, bei der der Normalvektor interpoliert wird, wird oft auch als „Phong-Shading“ bezeichnet, ist allerdings nicht zu verwechseln mit dem weiter oben beschriebenen Phong-Beleuchtungsmodell.

²Der Name kommt daher, dass die Gewichte λ_1, λ_2 und $1 - \lambda_1 - \lambda_2$ genau jenen Massen entsprechen, die man an den Eckpunkten anbringen müsste, um den Schwerpunkt des Dreiecks (das Baryzentrum) in den Punkt \mathbf{p} zu verlagern.

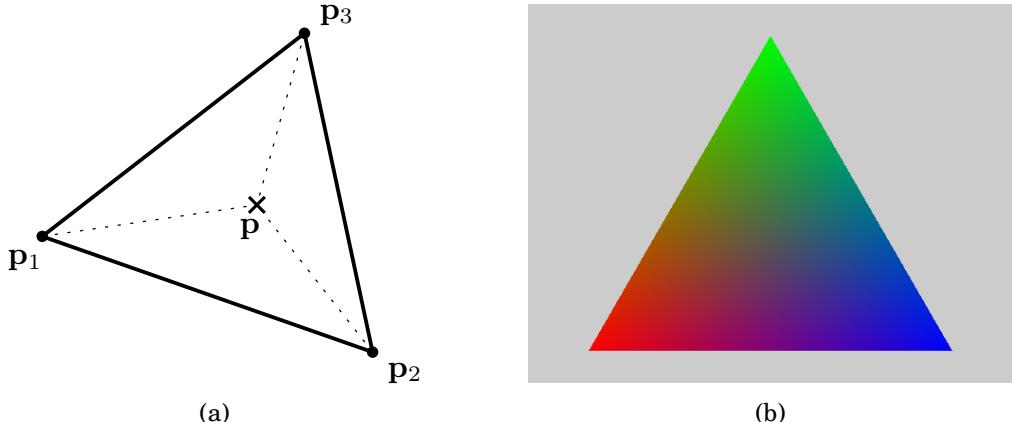


Abbildung 7: Baryzentrische Koordinaten. (a) jeder Punkt \mathbf{p} in der Ebene eines Dreiecks lässt sich in Baryzentrischen Koordinaten als gewichtete Summe der Eckpunkte $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ darstellen; die drei Baryzentrischen Koordinaten entsprechen den Verhältnissen der Flächen der zum Punkt \mathbf{p} aufgespannten Unterdreiecke zur Gesamtfläche. (b) lineare Interpolation von an den Eckpunkten definierten Farbwerten (hier Rot, Grün und Blau) über ein Dreieck.

Ebene des Dreiecks der linear interpolierte Wert

$$\alpha = (1 - \lambda_1 - \lambda_2) \cdot \alpha_1 + \lambda_1 \cdot \alpha_2 + \lambda_2 \cdot \alpha_3 \quad (4)$$

wobei λ_1 und λ_2 die Baryzentrischen Koordinaten von \mathbf{p} sind. Das Resultat einer solchen Interpolation ist in Abbildung 7b zu sehen.

1.2 Texture Sampling

Um ein Objekt mit einer Textur zu versehen, werden jedem Vertex Texturkoordinaten zugeordnet und, wie in Abbildung 8 dargestellt, auf diesem Wege Teile des Texturbildes auf die Dreiecke des Objektes abgebildet. Die Textur repräsentiert in der Regel den diffusen Reflexionskoeffizienten der Objektoberfläche. Um ein texturiertes Objekt zu rendern, muss daher in der Beleuchtungsberechnung an jedem Punkt der diffuse Reflexionskoeffizient aus dem Texturbild gelesen werden.

Die Texturkoordinaten eines beliebigen Punktes auf der Oberfläche lassen sich durch lineare Interpolation der Texturkoordinaten der Dreiecksvertices bestimmen. Texturbilder liegen in der Regel als Rastergrafiken vor und die Texturkoordinaten eines von einem Sichtstrahl getroffenen Oberflächenpunktes werden im Allgemeinen niemals exakt auf einen Texel des Texturbildes fallen. Der Prozess, in dem für einen beliebigen Punkt im Texturkoordinatenraum basierend auf dem Texturbild eine Farbe ermittelt wird, wird als *Texture Sampling* bezeichnet.

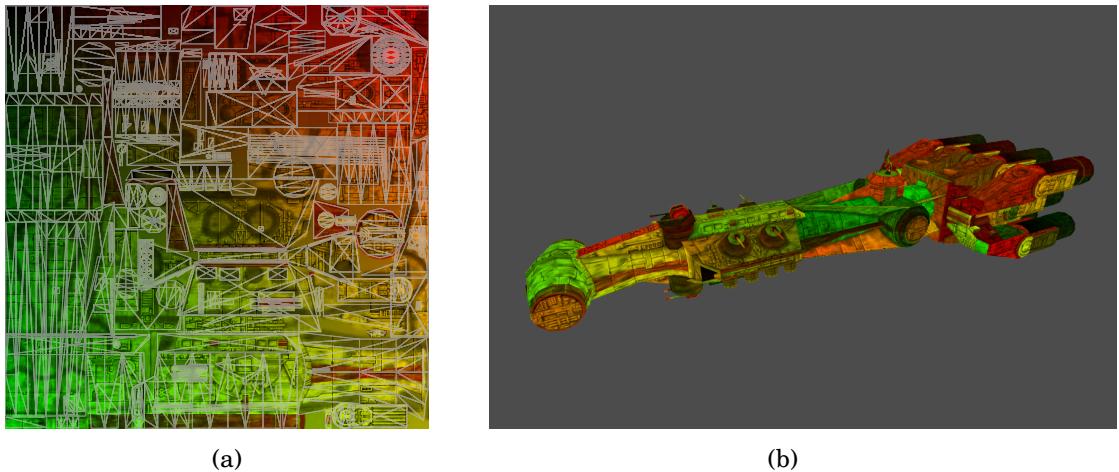


Abbildung 8: Texturemapping. (a) die Dreiecke eines Modells im Texturkoordinatenraum über dem Texturbild; (b) Rendering des mit dieser Textur bespannten Modells

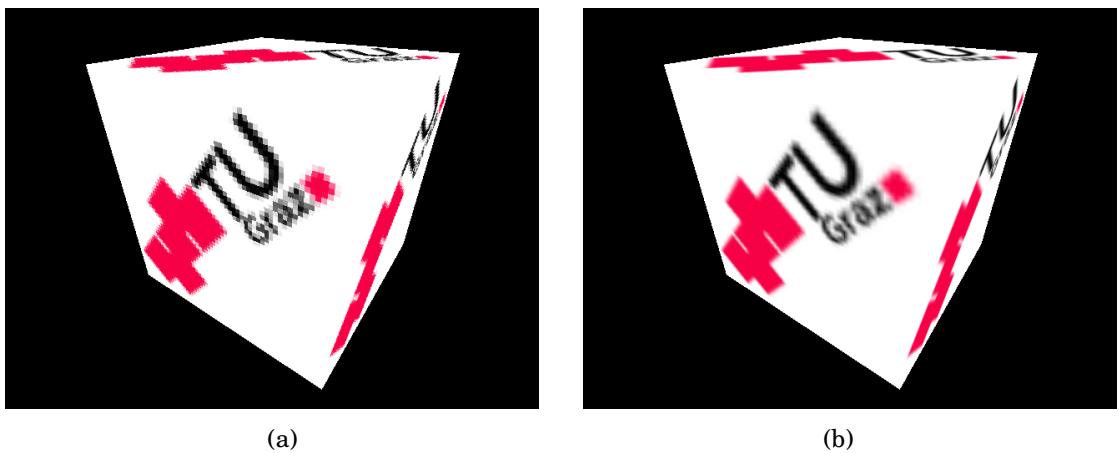


Abbildung 9: Vergleich zwischen (a) nearest-neighbor Sampling und (b) bilinearer Interpolation.

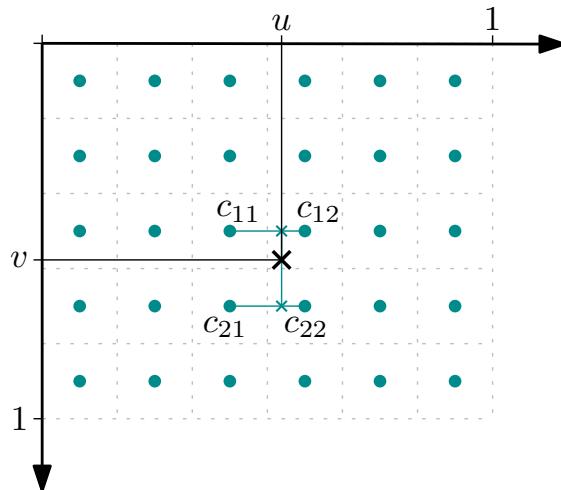


Abbildung 10: Bestimmung der Texturfarbe an einer beliebigen Sampleposition (u, v) durch bilineare Interpolation der Farben $c_{11}, c_{12}, c_{21}, c_{22}$ der umliegenden Texel. Im dargestellten Beispiel wird zuerst entlang der u -Achse zwischen c_{11} und c_{12} sowie zwischen c_{21} und c_{22} interpoliert und danach entlang der v -Achse zwischen den beiden Ergebnissen der ersten Interpolation, um die Farbe an der Sampleposition zu erhalten.

Eine sehr einfache Technik des Texture Sampling ist, die Farbe des der Sampleposition nächsten Texels zu verwenden, man spricht auch von *nearest-neighbor* Sampling. Wie in Abbildung 9 zu sehen, können Teile einer Textur infolge der Perspektive im Rendering stark vergrößert und andere Teile stark verkleinert erscheinen. Nearest-neighbor Sampling führt sehr schnell zu einem „verpixelten“ Aussehen in Bereichen starker Vergrößerung oder Verkleinerung. Das Ergebnis lässt sich verbessern, indem der Farbwert stattdessen durch bilineare Interpolation der umliegenden Texel gebildet wird.

Das Verfahren der bilinearen Interpolation ist in Abbildung 10 illustriert. Die Farbe an einer durch die Koordinaten (u, v) gegebenen Sampleposition wird aus den Farben $c_{11}, c_{12}, c_{21}, c_{22}$ der vier umliegenden Texel bestimmt. Dabei wird zuerst entlang einer Koordinatenachse zwischen den Farbwerten in der selben Zeile oder Spalte linear interpoliert und danach entlang der anderen Koordinatenachse zwischen den beiden Ergebnissen der ersten Interpolation nochmals linear interpoliert, um den finalen Farbwert für die Sampleposition zu erhalten. Es ist irrelevant, ob dabei zuerst entlang der u -Achse oder der v -Achse interpoliert wird, beide Varianten führen zum selben Ergebnis.

2 Aufgabe

Ziel dieser Übung ist es, den in task1a entwickelten, einfachen Raycaster um Phong-Shading zu erweitern. Wie bereits in task1a kümmert das zur Verfügung gestellte Frame-

work sich um das Einlesen einer Szenenbeschreibung sowie um die Ausgabe der Ergebnisse. Auch die Strahlgenerierung und Abtastung der Bildebene ist bereits im Framework implementiert. Ihre Aufgabe ist es, die einzelnen Schritte der Beleuchtungsberechnung in den «task1b.cpp» vordefinierten Funktionen auszuprogrammieren.

2.1 Interpolation der Oberflächenparameter (3 Punkte)

Um die Beleuchtung am Schnittpunkt eines Strahls mit einem Dreiecksnetz zu bestimmen, müssen zunächst wie in Abschnitt 1.1 beschrieben die Oberflächennormale und Texturkoordinaten für diesen Punkt bestimmt werden. Implementieren sie die Funktion

```
void computeSurfaceParameters(const Triangle* triangle,
                             const float2& barycentric,
                             float3& n,
                             float2& texCoords)
```

so, dass sie die Oberflächennormale n und Texturkoordinaten texCoords für den in seinen Baryzentrischen Koordinaten barycentric gegebenen Punkt auf dem Dreieck triangle durch lineare Interpolation der Oberflächennormalen und Textukoordinaten der Vertices berechnet.

2.2 Phong Shading (4 Punkte)

Sind die Oberflächen- und Materialparameter für einen Punkt bekannt, kann die lokale Beleuchtung der Oberfläche an diesem Punkt berechnet werden. Implementieren Sie die Funktion

```
float3 phong(const float3& p,
              const float3& n,
              const float3& v,
              const float3& k_d,
              const float3& k_s,
              float shininess,
              const float3& lightPos,
              const float3& lightColor)
```

so, dass sie, wie in Abschnitt 1 beschrieben, die Farbe der durch eine Punktlichtquelle beleuchteten Oberfläche an der Position p mit der Oberflächennormale n aus Blickrichtung v berechnet und zurückgibt. Position und Farbe der Punktlichquelle werden in den Parametern lightPos und lightColor übergeben. k_d und k_s sowie shininess enthalten die jeweiligen Materialparameter der Oberfläche, letzterer entspricht der in Gleichung (2) als m bezeichneten Glattheit der Oberfläche.

Die Funktion

```

float3 shade(const Scene1b& scene ,
             const float3& p ,
             const float3& n ,
             const float3& v ,
             const float3& k_d ,
             const float3& k_s ,
             float shininess ,
             int bounce ,
             int max_bounce)

```

soll dann die Farbe des in p gegebenen Punktes mit Oberflächennormale n bei Betrachtung aus Blickrichtung v unter Beleuchtung durch alle Lichtquellen der in $scene$ gegebenen Szene bestimmen, wobei k_d und k_s und $shininess$ wieder den Materialparametern entsprechen. Die Liste der Lichtquellen in der Szene ist über die Methode `scene.getLightList()` erreichbar. Die Parameter `bounce` und `max_bounce` werden erst im weiter unten beschriebenen Bonustask relevant.

Für jeden Primärstrahl wird vom Framework die Funktion

```

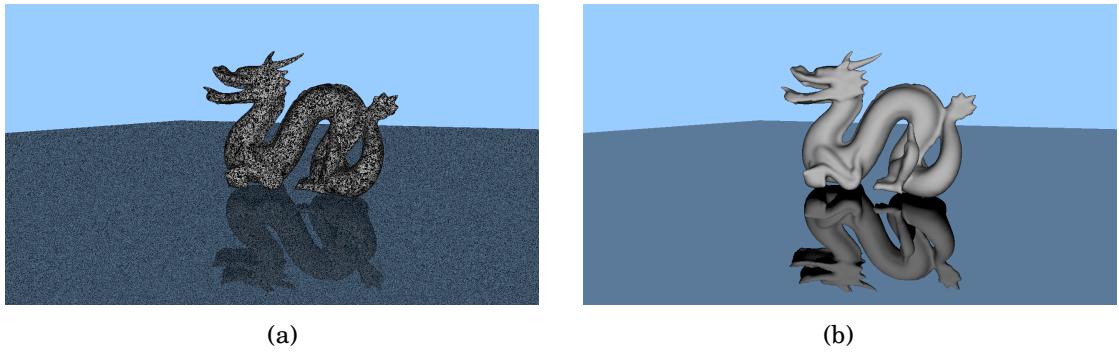
float3 traceRay(const Scene1b& scene ,
                 const Ray& ray ,
                 int bounce ,
                 int max_bounce)

```

aufgerufen, um die Farbe des entsprechenden Pixels des Ausgabebildes zu bestimmen. Implementieren Sie diese Funktion so, dass sie den im Parameter ray übergebenen Strahl gegen die Objekte der in $scene$ gegebenen Szene testet. Falls der Strahl auf ein Objekt trifft, soll die nach dem in Abschnitt 1 beschriebenen Phong-Shading Modell ermittelte Farbe des getroffenen Punktes auf der Objektoberfläche zurückgegeben werden. Falls der Strahl kein Objekt in der Szene trifft, soll die Szenenhintergrundfarbe zurückgegeben werden. Die Parameter `bounce` und `max_bounce` werden auch hier erst mit dem in Abschnitt 2.5 beschriebenen Bonustask relevant.

Aufgrund der begrenzten Genauigkeit von Gleitkommaoperationen, liegt der errechnete Schnittpunkt von Strahl und Dreieck im Allgemeinen nicht exakt in der Ebene des Dreiecks, sondern um einen kleinen Rundungsfehler davor oder dahinter, was zu Artefakten wie den in Abbildung 11 gezeigten führen kann. Verwenden Sie daher in allen Berechnungen nicht direkt den über die `scene.intersectWithRay()` Methode ermittelten Schnittpunkt mit der Objektoberfläche, sondern verschieben Sie diesen zunächst noch um die in der Konstante `epsilon` vorgegebene Distanz in Richtung der Flächennormale des getroffenen Dreiecks um sicherzustellen, dass die Koordinaten des Punktes, an dem die Beleuchtung bestimmt wird, nicht innerhalb des betrachteten Objektes liegen.

Um die Materialparameter k_d , k_s und m für einen Punkt auf der Oberfläche eines Objektes zu bestimmen, verwenden Sie die Methode `scene.materialParameters()`.



(a)

(b)

Abbildung 11: (a) Aufgrund von Gleitkommaengenauigkeiten in der Schnittpunktberechnung liegen die Punkte, an denen die Beleuchtung ausgewertet wird nicht exakt auf der Objektoberfläche, sondern teilweise um einen Rundungsfehler dahinter und damit im Schatten des eigenen Objektes, was zu einem stark verrauschten Rendering führen kann. (b) durch Verschieben des errechneten Schnittpunktes um eine kleine Distanz ϵ in Richtung der Dreiecksnormale wird sichergestellt, dass der Schnittpunkt auf oder außerhalb der Objektoberfläche liegt.

2.3 Texture Sampling (3 Punkte)

Um den diffusen Reflexionskoeffizienten k_d für einen Punkt auf der Oberfläche eines texturierten Objektes zu bestimmen, ruft das Framework die Funktion

```
float4 sampleTexture(const float2& texCoords,  
                     const surface<R8G8B8A8>& texture)
```

auf. Implementieren Sie diese Funktion so, dass sie die Farbe an der durch `texCoords` gegebenen Sampleposition wie in Abschnitt 1.2 beschrieben durch bilineare Interpolation aus dem in `texture` übergebenen Texturbild bestimmt und zurückgibt. Samplepositionen, die außerhalb des Texturbildes fallen, sollen auf den Rand des Bildes beschnitten werden.

2.4 Schatten (2 Punkte)

Ein einfacher Weg um festzustellen, ob ein Punkt im Schatten einer Lichtquelle liegt oder nicht, ist, einen Strahl von diesem Punkt aus zur Lichtquelle zu konstruieren und zu testen, ob dieser Strahl ein Objekt trifft oder nicht. Implementieren Sie die Funktion

```
bool inShadow(const float3& p,  
              const float3& lightPos,  
              const Scene1b& scene)
```

so, dass sie für den im Parameter `p` übergebenen Punkt feststellt, ob dieser in der in `scene` gegebenen Szene im Schatten einer an den Koordinaten `lightPos` befindlichen Punktlichtquelle liegt oder nicht und je nachdem `true` oder `false` returned. Modifizieren Sie weiters die Funktion `shade()` so, dass sie nur die Beleuchtung durch Lichtquellen, deren Licht den zu schattierenden Punkt auch erreicht zählt.

2.5 Bonus: Rekursives Raytracing (3 Punkte)

Modifizieren Sie die Funktionen `traceRay()` und `shade()` so, dass Oberflächen mit einem `shininess` Parameter von Unendlich als perfekt glatte Spiegeloberflächen behandelt werden. Der spekulare Anteil einer solchen Oberfläche wird bestimmt, indem ein weiterer Strahl entsprechend der Reflexion des auf die Oberfläche einfallenden Strahls um die Oberflächennormale ausgesandt wird. Die Farbe des fertig beleuchteten Oberflächenpunktes ergibt sich dann als Summe des diffusen Anteils nach Gleichung (1) und der Farbe des rekursiv ausgesandten, reflektierten Strahls gewichtet mit dem spekularen Reflexionskoeffizienten.

Verwenden Sie den Parameter `bounce` als Zähler für die aktuelle Rekursionstiefe, um eine endlose Rekursion (beispielsweise im Falle zweier einander zugewandter Spiegel) zu vermeiden. Im initialen Aufruf von `traceRay` durch das Framework wird `bounce` immer auf 0 initialisiert. Im Parameter `max_bounce` wird dabei maximale Rekursionstiefe, ab der abgebrochen werden soll, übergeben. Einem Strahl, der wegen Überschreiten der maximalen Rekursionstiefe abgebrochen wird, soll dabei die Szenenhintergrundfarbe zugewiesen werden.

3 Abgabe

Die Aufgaben bestehen jeweils aus mehreren Schritten, die zum Teil aufeinander aufbauen, jedoch unabhängig voneinander beurteilt werden. Dadurch ist einerseits eine objektive Beurteilung sichergestellt und andererseits gewährleistet, dass auch bei unvollständiger Lösung der Aufgaben Punkte erzielt werden können.

Wir weisen ausdrücklich darauf hin, dass die Übungsaufgaben von jedem Teilnehmer *eigenständig* gelöst werden müssen. Wenn Quellcode anderen Teilnehmern zugänglich gemacht wird (bewusst oder durch Vernachlässigung eines gewissen Mindestmaßes an Datensicherheit), wird das betreffende Beispiel bei allen Beteiligten mit 0 Punkten bewertet, unabhängig davon, wer den Code ursprünglich erstellt hat. Ebenso ist es nicht zulässig, Code aus dem Internet, aus Büchern oder aus anderen Quellen zu verwenden. Es erfolgt sowohl eine automatische als auch eine manuelle Überprüfung auf Plagiate.

Da die abgegebenen Programme halbautomatisch getestet werden, muss die Übergabe der Parameter mit Hilfe von entsprechenden Konfigurationsdateien genauso erfolgen wie bei den einzelnen Beispielen spezifiziert. Insbesondere ist eine interaktive Eingabe von Parametern nicht zulässig. Sollte aufgrund von Änderungen am Konfigurationssystem die Ausführung der abgegebenen Dateien mit den Testdaten fehlschlagen, wird das Beispiel mit 0 Punkten bewertet. Die Konfigurationsdateien liegen im JSON-Format vor, zu deren

Auswertung steht Ihnen die Klasse Config zur Verfügung. Die Verwendung der Klasse ist aus dem Programmgerüst ersichtlich.

Jede Konfigurationsdatei enthält zumindest einen Testfall und dessen Konfiguration. Es ist auch möglich, dass eine Konfigurationsdatei mehrere Testfälle enthält, um gemeinsame Parameter nicht mehrfach in verschiedenen Dateien spezifizieren zu müssen. In manchen Konfigurationsdateien finden sich auch einstellbare Parameter, die in Form eines select Feldes vorliegen. Diese sollen die Handhabung der Konfigurationsdateien erleichtern und ein einfaches Umschalten der Modi gewährleisten.

Es steht Ihnen frei, z.B. zu Testzwecken eigene Erweiterungen zu implementieren. Stellen Sie jedoch sicher, dass solche Erweiterungen in Ihrem abgegebenen Code deaktiviert sind, damit ein Vergleich der abgegebenen Arbeiten mit unserer Referenzimplementierung möglich ist.

Die Programmgerüste, die zur Verfügung gestellt werden, sind unmittelbar aus unserer Referenzimplementierung abgeleitet, indem nur jene Teile entfernt wurden, die dem Inhalt der Übung entsprechen. Die Verwendung dieser Gerüste ist nicht zwingend, aber Sie ersparen sich sehr viel Arbeit, wenn Sie davon Gebrauch machen.