5. Praktikum in Computergrafik und Bildverarbeitung

Lisa Obermaier, Simon Thum

Frist 25. Juni 2019, 23:59

Berechnen Sie die Normalenvektoren für jeden Ihrer Vertices. Berechnen Sie diese so, dass die Normalenvektoren normiert sind (d.h. Länge = 1) und in Richtung der Winkelhalbierenden der zwei oder drei Flächennormalenvektoren der angrenzenden Flächen zeigen. Bauen Sie eine Fallunterscheidung in Ihren Algorithmus ein, der ab einem bestimmten Grenzwinkel (möglichst GUI gesteuert einstellbar) zwischen den beiden Flächennormalenvektoren aneinander grenzender Flächen die Winkelhalbierung unterlässt.

Die Zylinderpunkte werden unverändert in einer Schleife berechnet:

```
int i = 0;
for (float angle = 0.0f; angle < (2.0f*GL_PI); angle += (GL_PI / (2 * tesselierung)))
{
    // Berechne x und y Positionen des naechsten Vertex
    float x = .5f*sin(angle);
    float y = .5f*cos(angle);

    // Spezifiziere den naechsten Vertex des Triangle_Fans
    m3dLoadVector3(bodenVertices[i], x, y, 0);
    m3dLoadVector3(boden2Vertices[i], x, y, 1);

    //printf("Schleife i: %d, Wert x: %f, Wert y: %f\n\n", i, x, y);
    i++;
}
```

Beim Befüllen der *GLBatches* werden allerdings zugleich Normalvektoren angelegt. Für Deckel und Boden des Zylinders stehen diese einfach senkrecht auf der Kreisfläche (Z-Komponente 1/- 1).

```
for (i = 0; i < 7; i++) {
    geometryBatch.Normal3f(0, 0, 1);
    geometryBatch.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 1.0);
    geometryBatch.Normal3f(0, 0, 1);
    geometryBatch.Vertex3f(0.0, 0.0, 1.0);
    geometryBatch.Normal3f(0, 0, 1);
    geometryBatch.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);

    normVecs.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i][0] + 0, boden2Vertices[i][1] + 0, 1.0 + 1);
    normVecs.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i + 1][0] + 0, boden2Vertices[i + 1][1] + 0, 1.0 + 1);
}
```

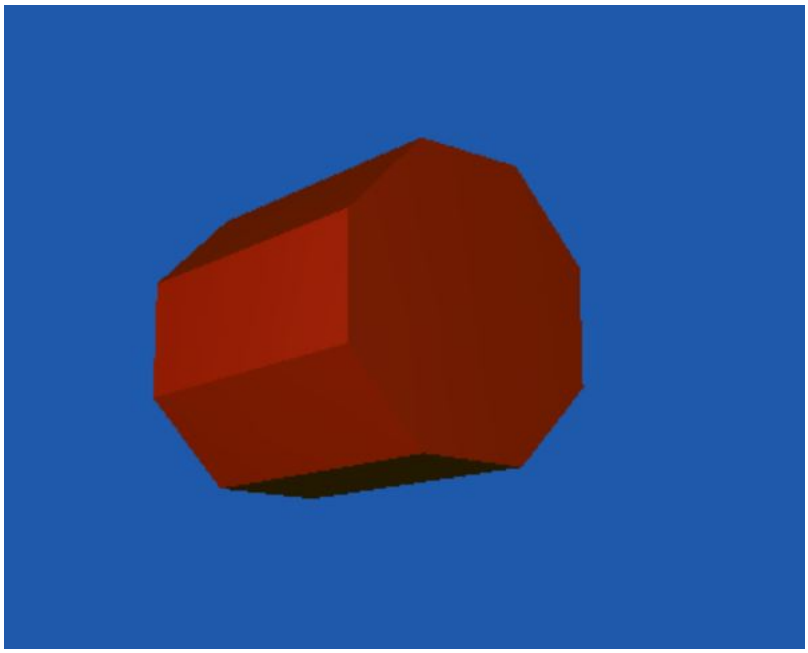
```
for (i = 0; i < 7; i++) {
    geometryBatch.Normal3f(0, 0, -1);
    geometryBatch.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 0.0);
    geometryBatch.Normal3f(0, 0, -1);
    geometryBatch.Vertex3f(0.0, 0.0, 0.0);
    geometryBatch.Normal3f(0, 0, -1);
    geometryBatch.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 0.0);

    normVecs.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 0.0);
    normVecs.Vertex3f(boden2Vertices[i][0] + 0, boden2Vertices[i][1] + 0, 0.0 - 1);
    normVecs.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 0.0);
    normVecs.Vertex3f(boden2Vertices[i + 1][0] + 0, boden2Vertices[i + 1][1] + 0, 0.0 - 1);
}
```

Gleichzeitig wird der Batch *normVecs* mit den berechneten Punkten und den Punkten in komponentenweiser Addition mit dem Normalvektoren gefüllt, diese Punkte werden in der nächsten Teilaufgabe benutzt, um die Normalvektoren sichtbar zu machen.

Der Zylindermantel ist etwas komplexer, da jeweils sechs Vertexpunkte die zwei Dreiecke einer Mantelwand ausmachen. Damit die Normalvektoren senkrecht zur berechneten Fläche stehen, müssen von jedem Punkt aus die beiden Vektoren zu den Nachbarpunkten (nächster Punkt im aktuellen Kreis, und gegenüberliegender Punkt in der anderen Kreisscheibe des Zylinders) als Kreuzprodukt genommen. Das Ergebnis wird als Normalvektor für den Punkt geladen und für die spätere Darstellung auf den *normVecs*-Punkt addiert.

```
for (i = 0; i < 7; i++) {
    m3dLoadVector3(vecU, boden2Vertices[i + 1][0] - bodenVertices[i][0], boden2Vertices[i + 1][1] - bodenVertices[i][1], 1.0 - 0.0);
    m3dLoadVector3(vecV, boden2Vertices[i + 1][0] - boden2Vertices[i][0], boden2Vertices[i + 1][1] - boden2Vertices[i][1], 1.0 - 1.0);
    m3dCrossProduct3(kreuzprod, vecU, vecV);
    m3dNormalizeVector3(kreuzprod);

    //PUNKT 0
    geometryBatch.Normal3f(kreuzprod[0], kreuzprod[1], kreuzprod[2]);
    geometryBatch.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i][0] + kreuzprod[0], boden2Vertices[i][1] + kreuzprod[1], 1.0 + kreuzprod[2]);

    //PUNKT 1
    geometryBatch.Normal3f(kreuzprod[0], kreuzprod[1], kreuzprod[2]);
    geometryBatch.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
    normVecs.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
    normVecs.Vertex3f(bodenVertices[i][0] + kreuzprod[0], bodenVertices[i][1] + kreuzprod[1], 0.0 + kreuzprod[2]);

    //PUNKT 2
    geometryBatch.Normal3f(kreuzprod[0], kreuzprod[1], kreuzprod[2]);
    geometryBatch.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i + 1][0] + kreuzprod[0], boden2Vertices[i + 1][1] + kreuzprod[1], 1.0 + kreuzprod[2]);

    //PUNKT 3
    geometryBatch.Normal3f(kreuzprod[0], kreuzprod[1], kreuzprod[2]);
    geometryBatch.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
    normVecs.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
    normVecs.Vertex3f(bodenVertices[i][0] + kreuzprod[0], bodenVertices[i][1] + kreuzprod[1], 0.0 + kreuzprod[2]);

    //PUNKT 4
    geometryBatch.Normal3f(kreuzprod[0], kreuzprod[1], kreuzprod[2]);
    geometryBatch.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
    normVecs.Vertex3f(boden2Vertices[i + 1][0] + kreuzprod[0], boden2Vertices[i + 1][1] + kreuzprod[1], 1.0 + kreuzprod[2]);

    //PUNKT 5
    geometryBatch.Normal3f(kreuzprod[0], kreuzprod[1], kreuzprod[2]);
    geometryBatch.Vertex3f(bodenVertices[i + 1][0], bodenVertices[i + 1][1], 0.0);
    normVecs.Vertex3f(bodenVertices[i + 1][0], bodenVertices[i + 1][1], 0.0);
    normVecs.Vertex3f(bodenVertices[i + 1][0] + kreuzprod[0], bodenVertices[i + 1][1] + kreuzprod[1], 0.0 + kreuzprod[2]);
}
```

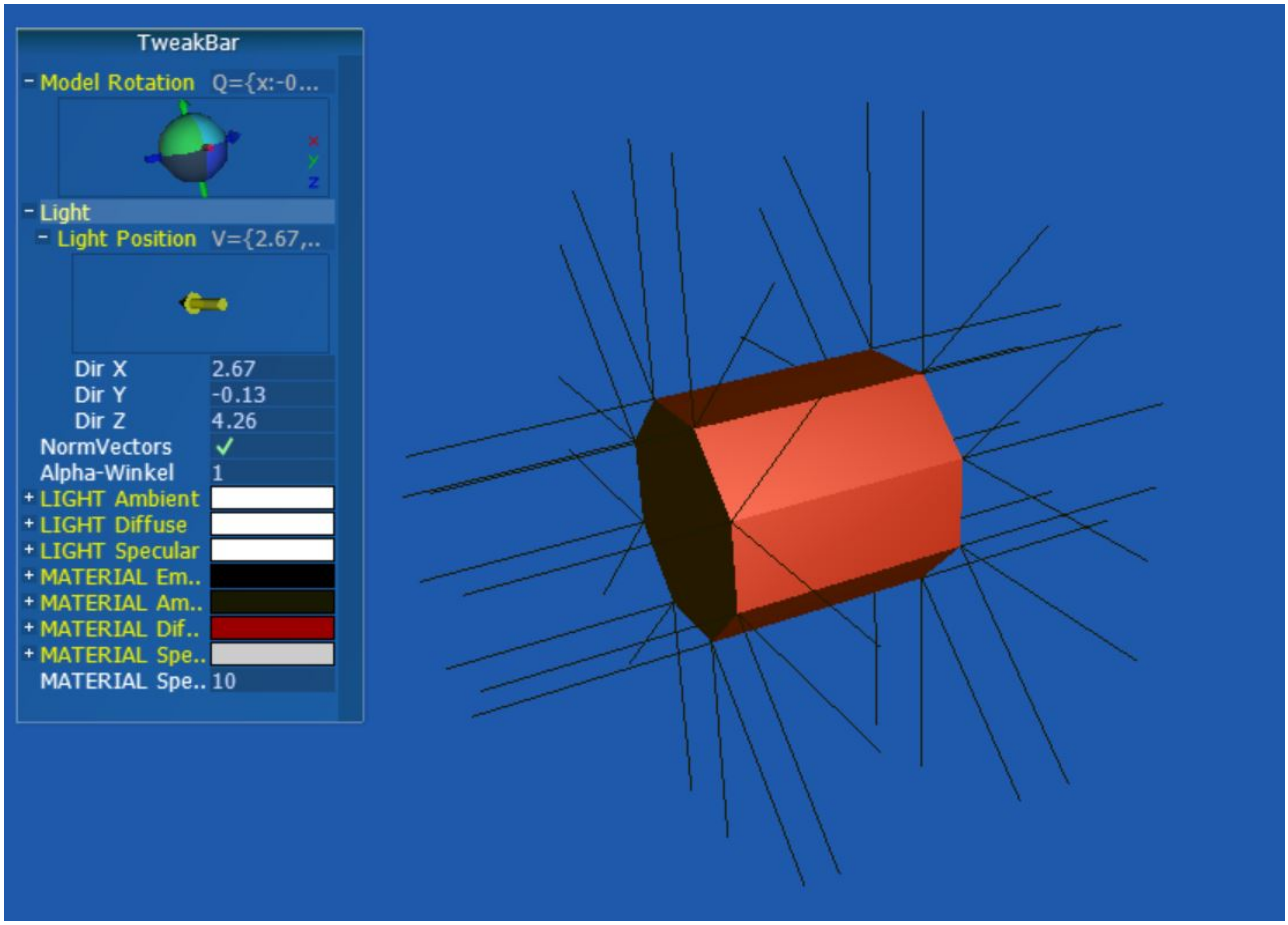
Ergebnis: Harte Kanten an den Wandstücken der Mantelfläche.



Machen Sie die Normalenvektoren sichtbar, indem Sie für jeden Normalenvektor eine Linie zeichnen. Bauen Sie einen Schalter in das GUI Ihrer Applikation ein, dass Sie die Normalenvektoren ein- und ausblenden können.

Die bereits berechneten Normalenvektoren im GLBatch können nach Integration eines UI-Schalters einfach über *normVecs.Draw()* in *RenderScene()* gezeichnet werden.

Ergebnis: Jeder Eckpunkt besitzt drei Normalvektoren, je einen zu jeder angrenzenden Fläche. Dies bedingt die zuvor erwähnten harten Kanten.

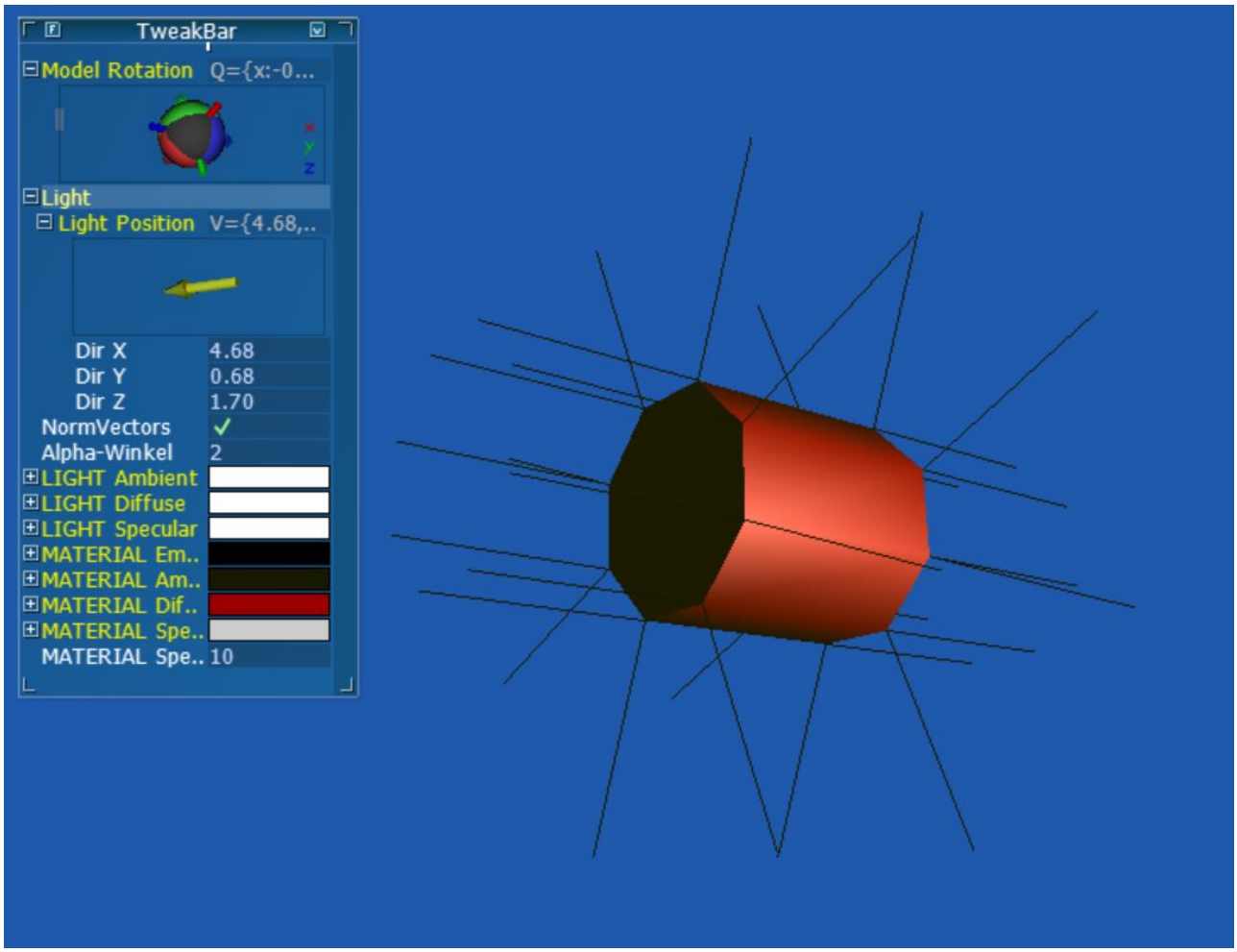


Durch einen Schalter in der GUI können zwei weitere Normalvektor-Modi gewählt werden. Modus 2 vereinheitlicht pro Vertexpunkt je zwei der Normalvektoren des Mantels durch Kompontenaddition in X- und Y-Richtung. Die Z-Richtung für den Normalvektor bleibt bei 0, da die Vektoren für die beiden Zylinderböden weiterhin konstant sind. Diese werden auch unverändert berechnet.

Als Auszug soll die Berechnung von drei Mantelpunkt-Normalvektoren berechnen. Der Vektor *lightVec* hält dabei aus Gründen der Übersichtlichkeit den normierten Normalvektor.

```
//PUNKT 0
m3dLoadVector3(lightVec, boden2Vertices[i][0], boden2Vertices[i][1], 0.0);
m3dNormalizeVector3(lightVec);

geometryBatch.Normal3f(lightVec[0], lightVec[1], lightVec[2]);
geometryBatch.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 1.0);
normVecs.Vertex3f(boden2Vertices[i][0], boden2Vertices[i][1], 1.0);
normVecs.Vertex3f(boden2Vertices[i][0] + lightVec[0], boden2Vertices[i][1] + lightVec[1], 1.0 + lightVec[2]);

//PUNKT 1
m3dLoadVector3(lightVec, bodenVertices[i][0], bodenVertices[i][1], 0.0);
m3dNormalizeVector3(lightVec);

geometryBatch.Normal3f(lightVec[0], lightVec[1], lightVec[2]);
geometryBatch.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
normVecs.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
normVecs.Vertex3f(bodenVertices[i][0] + lightVec[0], bodenVertices[i][1] + lightVec[1], 0.0 + lightVec[2]);

//PUNKT 2
m3dLoadVector3(lightVec, boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 0.0);
m3dNormalizeVector3(lightVec);

geometryBatch.Normal3f(lightVec[0], lightVec[1], lightVec[2]);
geometryBatch.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
normVecs.Vertex3f(boden2Vertices[i + 1][0], boden2Vertices[i + 1][1], 1.0);
normVecs.Vertex3f(boden2Vertices[i + 1][0] + lightVec[0], boden2Vertices[i + 1][1] + lightVec[1], 1.0 + lightVec[2]);
```

Ergebnis: Die Mantelfläche sieht in der Beleuchtung keine Kantenübergänge mehr zueinander, wohl jedoch zu den Böden.



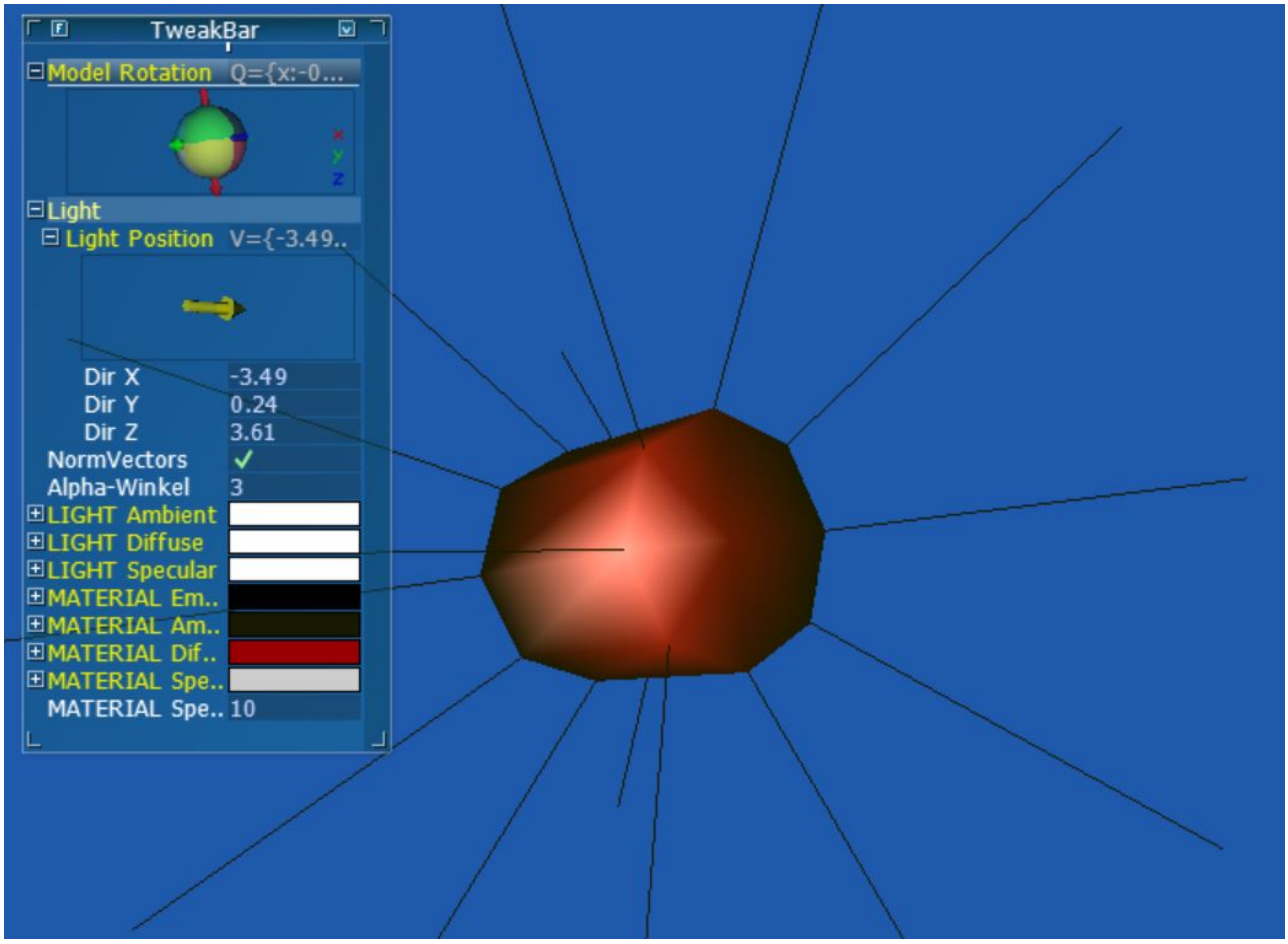
In Fall *3* wird nun zu jedem Vertexpunkt zusätzlich noch der Normalvektor des zugehörigen Bodens addiert und normiert.

Als Auszug soll aus Platzgründen die Berechnung eines Punktes genügen:

```
//Punkt 1
m3dLoadVector3(lightVec, bodenVertices[i][0], bodenVertices[i][1], 0.0);
m3dNormalizeVector3(lightVec);
m3dLoadVector3(vec1, bodenVertices[i][0] + lightVec[0], bodenVertices[i][1] + lightVec[1], 0.0 + lightVec[2]);
m3dLoadVector3(vec2, bodenVertices[i][0] + 0, bodenVertices[i][1] + 0, 0.0 - 1);
m3dAddVectors3(add, vec1, vec2);
normVecs.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
normVecs.Vertex3f(add[0], add[1], add[2]);

geometryBatch.Normal3f(add[0], add[1], add[2]);
geometryBatch.Vertex3f(bodenVertices[i][0], bodenVertices[i][1], 0.0);
```

Die Normalvektoren müssen nun bereits bei Berechnung der Bodenpunkte komplett berechnet werden. Dies geschieht durch die Vektoraddition der normalisierten Vektoren von Zylindermantel und Boden. Das Ergebnis liegt im Vektor *add*.

Ergebnis: Auch die Kanten zwischen Mantel und Boden verschmelzen, da jeder Vertexpunkt nur noch über einen Normalvektor verfügt, der die normalisierte Addition aller drei Komponenten darstellt.

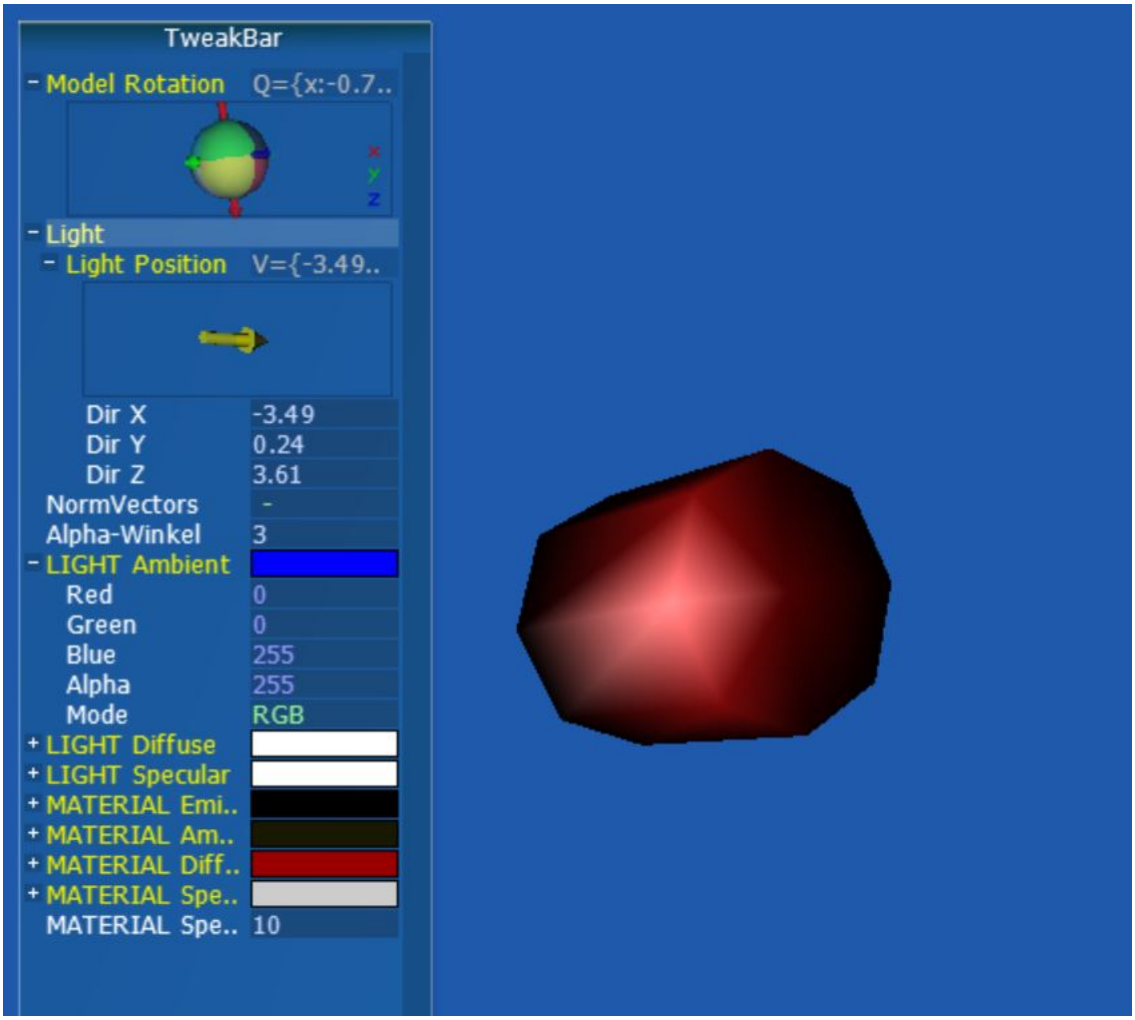


Ordnen Sie den Graphik-Primitiven (GL_TRIANGLES) des Zylinders Materialeigenschaften zu, d.h. Kombinationen von emissiven, ambienten, diffusen und spekularen (Farbe + Shininess) Anteilen (einstellbar über das GUI).

Definieren Sie eine unendlich entfernte Lichtquelle, bei der Sie Position, ambiente, diffuse und spekulare Lichtanteile (über das GUI) einstellen können. Beschreiben Sie die Wirkung der verschiedenen Lichtanteile bei der Beleuchtung, sowie die von Positionsänderungen der Lichtquelle.

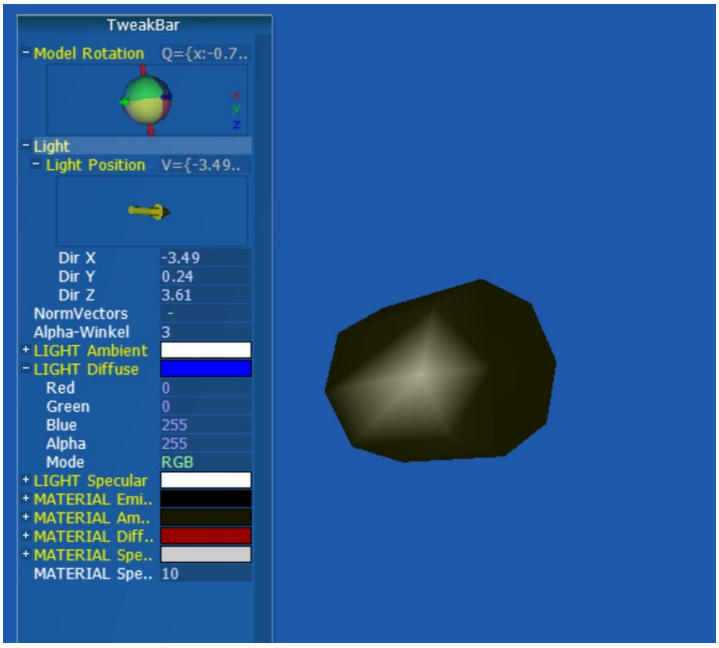
Ambient Light setzt die Hintergrundbelechtung, die ihre Farbe schwach und indirekt auf das Objekt abgibt.

Beispiel: Roter Zylinder, blaues *Ambient Light*. Leichte Violett-Tönung.



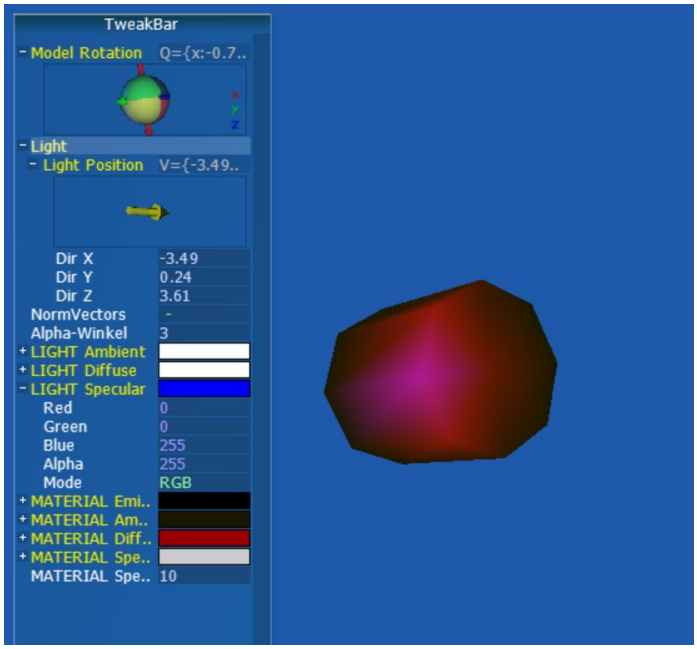
Diffuse Light überträgt die Farbe stark und direkt auf das gesamte Objekt, unabhängig von der Position der Lichtquelle.

Beispiel: Roter Zylinder, blaues *Diffuse Light*. Starke Farbvermengung in einen Braunton.



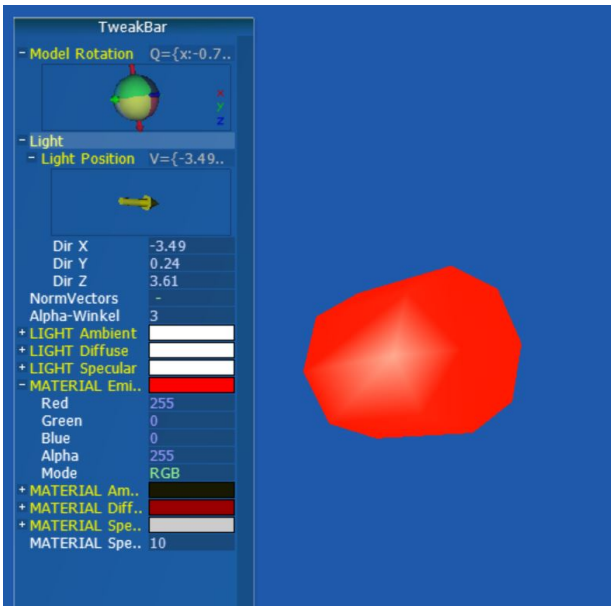
Specular Light bestimmt die Farbe des Highlights stark und direkt.

Beispiel: Roter Zylinder, blaues *Specular Light*. Das Highlight erscheint violett.



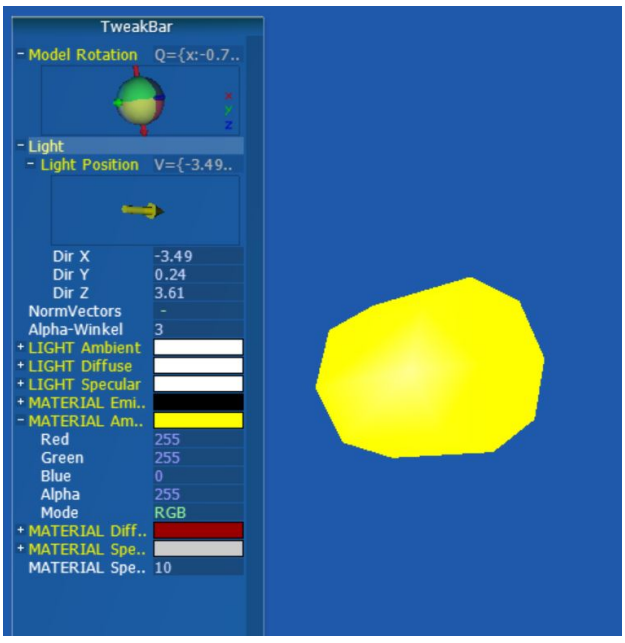
Emissive Material ist das, welches das Material selbst ausstrahlt, ergo “von innen heraus” selbst leuchtet.

Beispiel: Rotes *Emissive Material* lässt die Zylinderfläche selbst rot leuchten, Schattenwurf durch externe Lichtquelle findet nicht statt.



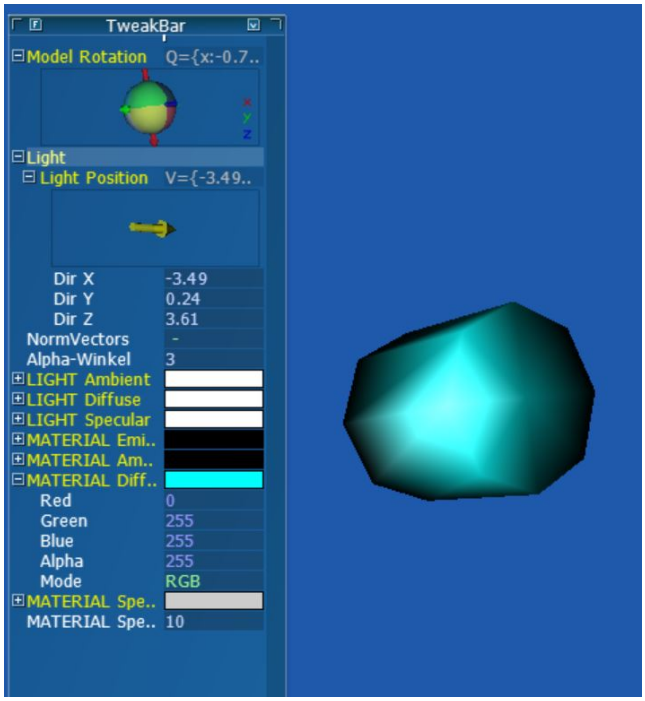
Ambient Material des Materials ist ähnlich *Emissive*, beleuchtet aber keine anderen Objekte um sich herum.

Beispiel: Gelbes *Ambient Material* lässt die Zylinderfläche aus sich selbst heraus einheitlich gelb erscheinen.



Diffuse Material muss für jeden Vertex berechnet werden und ist daher aufwändiger als das einheitliche *Ambient Material*.

Beispiel: Cyanfarbenes *Diffuse Material* färbt das Material hellblau, lässt aber Schwattenwurf durch die externe Lichtquelle weiterhin zu.



Das *Specular Material* bestimmt die Farbe der durch *Specular Light* beleuchteten Bildpunkte.

Beispiel: Selbst bei weißem *Specular Light* erscheint das Highlight violett, da der rote Zylinder blaues *Specular Material* erhalten hat.