

# Computergrafik.Online

## Drehbuch Shader

Hochschule Furtwangen University

Fakultät Digitale Medien

Betreut von:

Prof. Jirka Dell'Oro-Friedl

Version: 2.3


Letzte Änderung: 05.12.2018

Autor: Steven Romanek

# Inhalt

10.1 (A) Einleitung.....	2
10.2 (A) Vertex Shader .....	4
10.3 (I) Vertex Shader .....	5
10.4 (A) Geometry-Shader .....	6
10.5 (A) Pixel- / Fragment-Shader.....	8
10.6 (A) Flat-Shading.....	10
10.7 (A) Gouraud-Shading.....	12
10.8 (A) Phong-Shading.....	14
10.9 (I) Vergleich zwischen Flat-, Gouraud - und Phong-Shading.....	16
10.10 (A) Toon-/Cel-Shading.....	17
10.11 (A) Programmierung eines Shaders .....	19

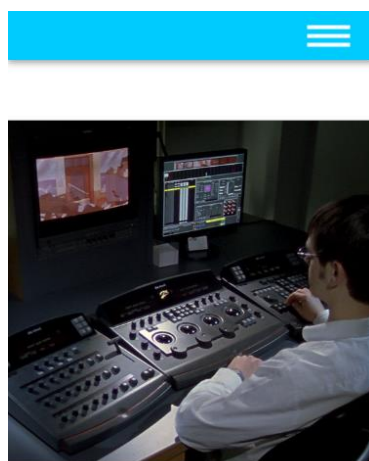
## 10.1 (A) Einleitung

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>100101</b> Shader sind spezielle Programme, welche in Computerspielen, in der Postproduktion von Videoinhalten und bei Computer Generated Imagery, kurz CGI, zum Einsatz kommen. Shader waren ursprünglich, wie der Name schon sagt, für das Schattieren bzw. Erzeugen von verschiedenen Stufen von Licht, Dunkelheit und Farben in der Computergrafik zuständig. Beispiele hierfür sind Flat-, Gouraud - und Phong-Shading, welche fest im Grafikchip verbaut sind. Heutzutage können Shader frei programmiert werden und übernehmen auch Aufgaben, die nichts mit dem ursprünglichen Schattieren zu tun haben, wie z.B. das Erzeugen neuer Geometrien.</p> <p><b>100102</b> Die Verarbeitung von Shadern findet in den sogenannten Shadereinheiten statt. Diese befinden sich in Grafikchips von Grafikkarten, welche ein wichtiger Bestandteil in Computern, Spielekonsolen, Smartphones und anderen vergleichbaren Geräten sind. In Grafikkarten gibt es eine sogenannte Grafikpipeline, welche die Reihenfolge der auszuführenden Shader festlegt.</p>	<p><b>100101</b></p> <ul style="list-style-type: none"> <li>- Sind spezielle Programme</li> <li>- Einsatz in Computerspielen, Postproduktion und bei CGI</li> <li>- Ursprünglich nur für Schatten zuständig</li> </ul> <p><b>100102</b></p> <ul style="list-style-type: none"> <li>- Shader werden in Shadereinheiten verarbeitet</li> <li>- Grafikkarten sind Bestandteil von Computern, Smartphones und anderen Geräten</li> <li>- Schnittstellen wie DirectX und OpenGL nötig</li> </ul>	<p><b>100101</b> Es werden nach und nach die Beispiele der Einsatzzwecke in Form von Bildern eingeblendet.</p> <p><b>100102</b> Nach dem Ausblenden der Einsatzzwecke, wird als erstes eine normale Grafikkarte eingeblendet und mit einem Pfeil gezeigt, wo sich darauf der Grafikchip befindet. Danach wird das Bild der Grafikkarte verkleinert und darunter erscheinen drei Geräte (PC, Smartphone, Spielekonsole). Daraufhin werden symbolisch drei Grafikkarten in diese</p>

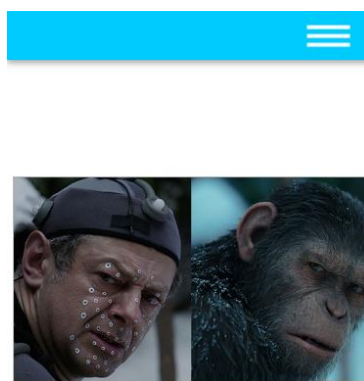
	Zur Nutzung von Shadern ist noch eine programmierbare Schnittstelle nötig, wie z.B. DirectX oder OpenGL.		Geräte geschoben, um hervorzuheben, dass diese Bestandteil solcher Geräte sind.
--	--	--	---



Text einblenden



Text einblenden



Text einblenden

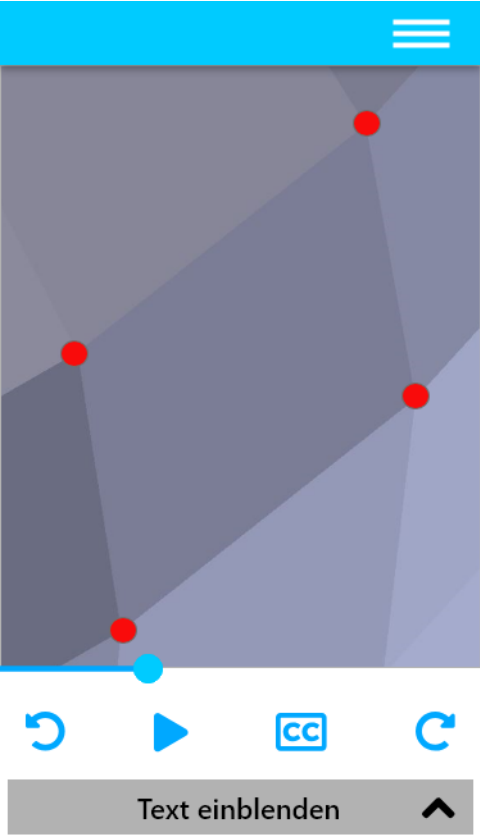


Text einblenden




Text einblenden

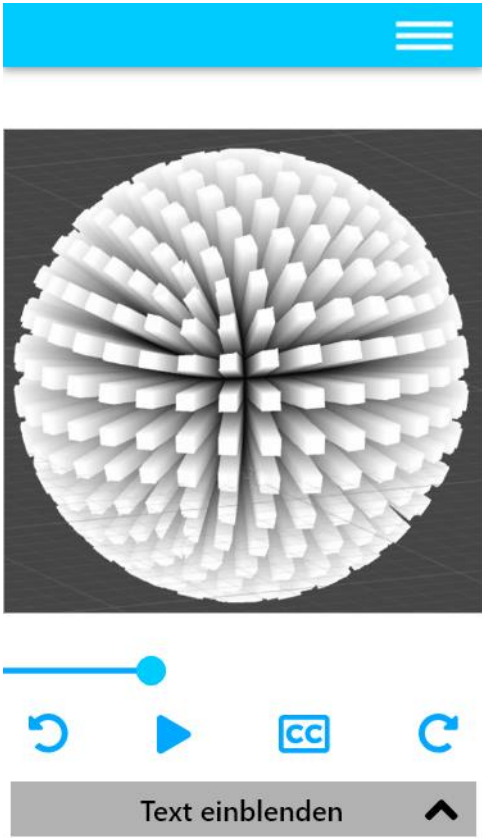
## 10.2 (A) Vertex Shader

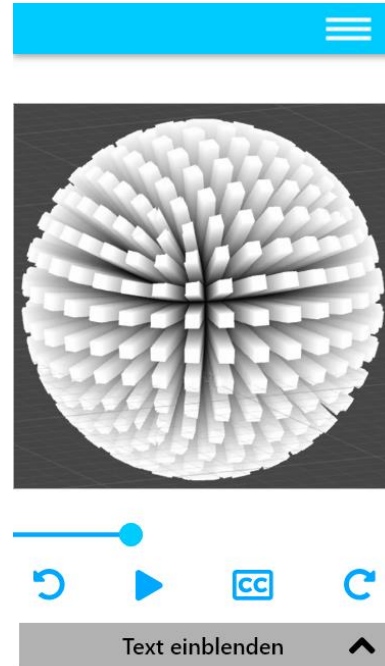
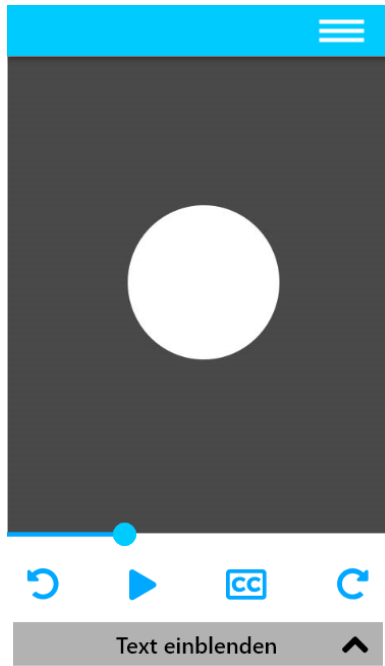
Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>100201</b> Vertex-Shader sind Programme, welche im Verlauf der Grafikpipeline in den Shadereinheiten einer Grafikkarte ausgeführt werden. Diese verarbeiten die sogenannten Vertices, bei denen es sich um Eckpunkte eines 3D-Modells handelt.</p> <p>Beim Vertex-Shader werden die Koordinaten der einzelnen Vertices im dreidimensionalen Raum für die zweidimensionale Darstellung transformiert. Dadurch lässt sich die Geometrie und somit die Form von Objekten beeinflussen, was sich wiederum auch auf die Beleuchtung auswirken kann. Da der Shader aber pro Vertex aufgerufen wird, kann er keine neuen Punkte zum 3D-Modell hinzufügen.</p>	<p><b>100201</b></p> <ul style="list-style-type: none"> <li>- Shader verarbeitet Vertices</li> <li>- Einfluss auf Geometrie und somit auch Beleuchtung</li> </ul>	<p><b>100201</b> Es wird in einen Ausschnitt des Roboters hineingezoomt. Dort werden nach und nach die Vertices markiert.</p>

## 10.3 (I) Vertex Shader

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
 <p>The screenshot shows a 3D rendered landscape with rolling hills in shades of blue and grey. At the top, there is a blue header bar with a white menu icon. On the right side of the landscape, there is a small blue 'i' icon. At the bottom, there is a slider control labeled 'Höhe der Vertices' with a blue knob positioned at the start, indicating 0%.</p>	<p><b>100301</b>          Hier ist ein Vertex Shader aktiv, welcher durch verschieben des Reglers Einfluss auf die Geometrie der Wellen nimmt.</p>	<p>-</p>	<p><b>100301</b>          Der User kann den Regler verschieben, wodurch sich die Höhe der Vertices und somit auch die Geometrie der Wellen verändert.</p>

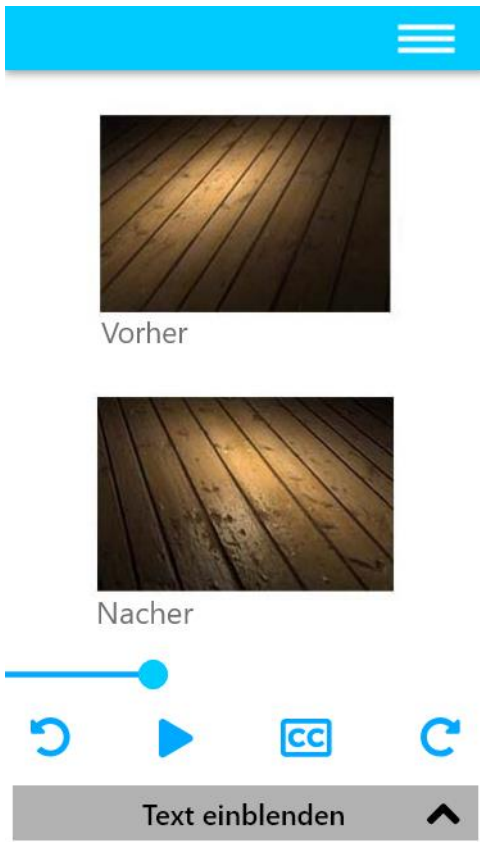
## 10.4 (A) Geometry-Shader

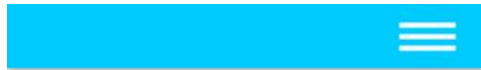
Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>100401</b> Der Geometry-Shader wird in der Grafikpipeline nach dem Vertex Shader aufgerufen, um neue primitive Geometrien aus bereits vorhandenen Punkten, Linien und Dreiecken zu erzeugen und diese erneut in die Grafikpipeline einzufügen.</p> <p>Beispiele für die Anwendung des Geometry-Shader sind die Erzeugung von Schattenvolumen oder die Erzeugung von Fell- oder Haargeometrie.</p> <p>Um Haare zu erzeugen, erhält der Geometry-Shader die benötigten Vertices vom Vertex-Shader als Input. Diese Vertices werden dann durch mehrere Kopien ersetzt, wodurch eine Haar-Struktur entsteht. Nach der Verarbeitung werden die fertigen Fragmente an den Pixel-Shader weitergegeben.</p>	<p><b>100401</b> - Erzeugung neuer Geometrien - z.B. Schatten und Haare</p>	<p><b>100401</b> Mithilfe der Vertices wird beispielhaft eine Haargeometrie erzeugt. Als erstes wird das Objekt mit glatter Oberfläche gezeigt, welche sich dann zu einer einfachen Haar-Struktur entwickelt.</p>





## 10.5 (A) Pixel- / Fragment-Shader

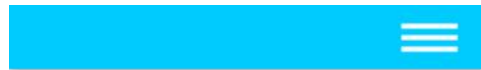
Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
 <p>Vorher</p> <p>Nachher</p> <p>Text einblenden</p>	<p><b>100501</b> Pixel-Shader, auch Fragment-Shader genannt, werden ebenfalls in den Shadereinheiten einer Grafikkarte ausgeführt. Der Shader ist für die Farbberchnung der einzelnen Pixel, auch Fragmente genannt, zuständig und folgt in der Grafikpipeline auf den Geometry-Shader.</p> <p>Pixel-Shader werden genutzt, um eine realistische Darstellung von Oberflächen- und Materialeigenschaften zu erreichen oder eine Texturdarstellung zu bewerkstelligen bzw. zu verändern. Ein Pixel kann dabei aus mehreren Fragmenten bestehen, was zum Beispiel bei Transparenz der Fall ist, wenn ein Objekt hinter einer Scheibe steht. Die richtige Farbe wird über eine Beleuchtungsberechnung zugewiesen.</p>	<p><b>100501</b> - Darstellung realistischer Oberflächen- und Materialeigenschaften</p>	<p><b>100501</b> Die Anwendung des Pixel-Shaders wird beispielhaft an einem Bild gezeigt, sodass man sieht was sich durch den Pixel-Shader verändert.</p>



Timeline slider with a blue dot at approximately 25%.

Navigation icons: a blue circular arrow, a blue play button, a blue 'CC' icon, and a blue circular arrow.

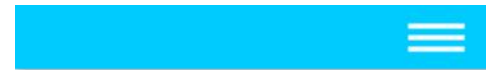
Text einblenden ^



Timeline slider with a blue dot at approximately 25%.

Navigation icons: a blue circular arrow, a blue play button, a blue 'CC' icon, and a blue circular arrow.

Text einblenden ^



Vorher



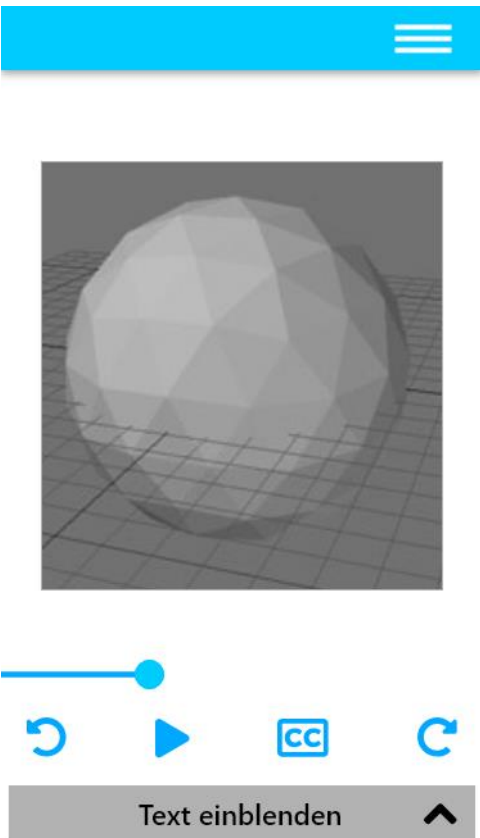
Nacher

Timeline slider with a blue dot at approximately 25%.

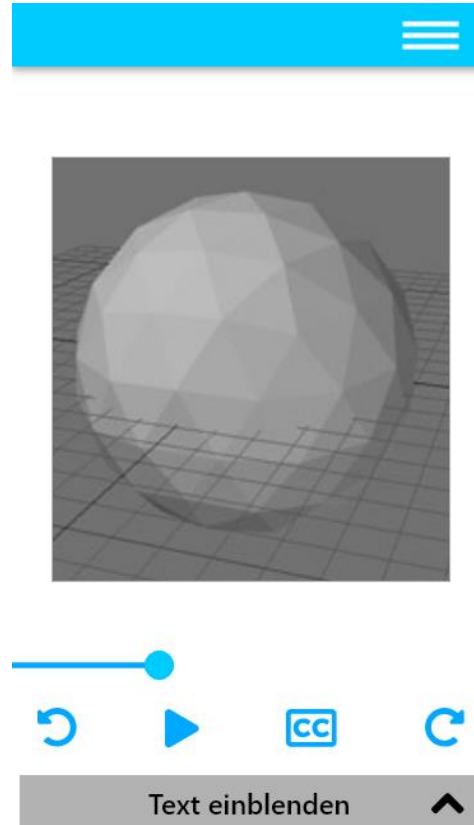
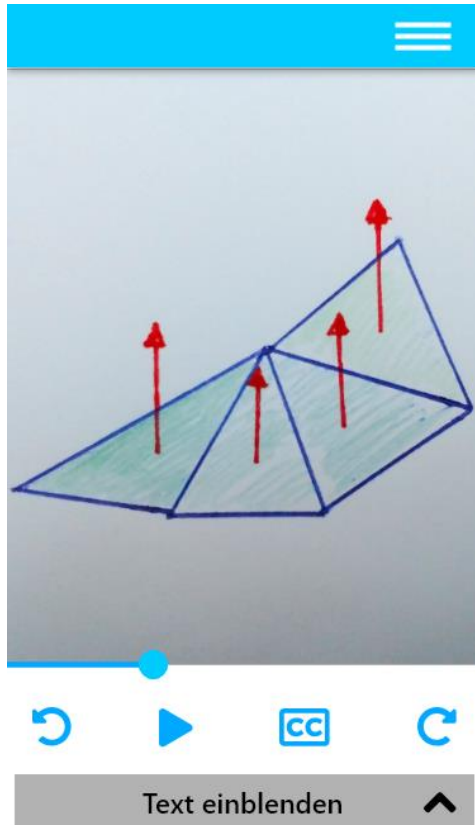
Navigation icons: a blue circular arrow, a blue play button, a blue 'CC' icon, and a blue circular arrow.

Text einblenden ^

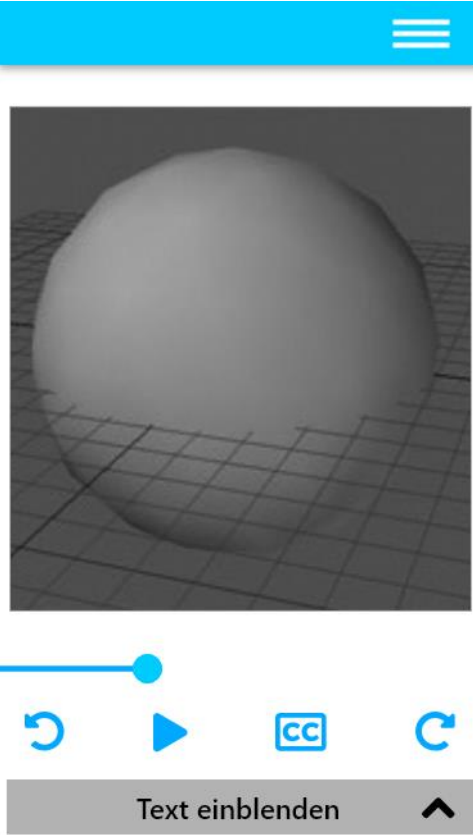
## 10.6 (A) Flat-Shading

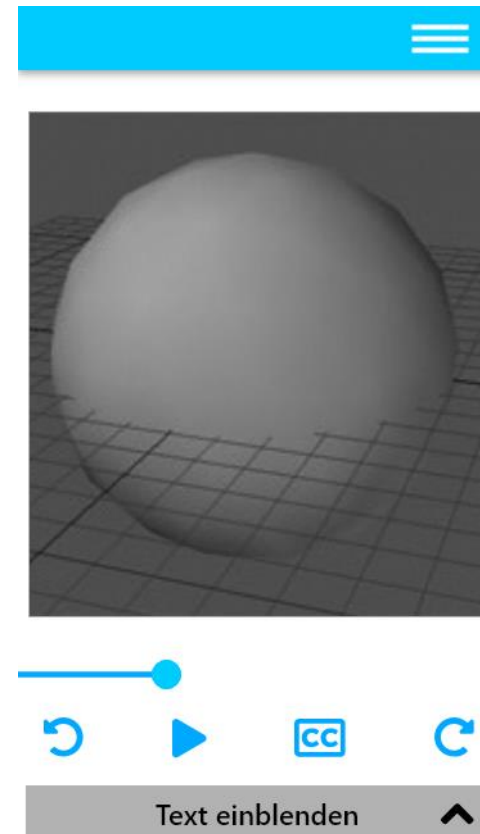
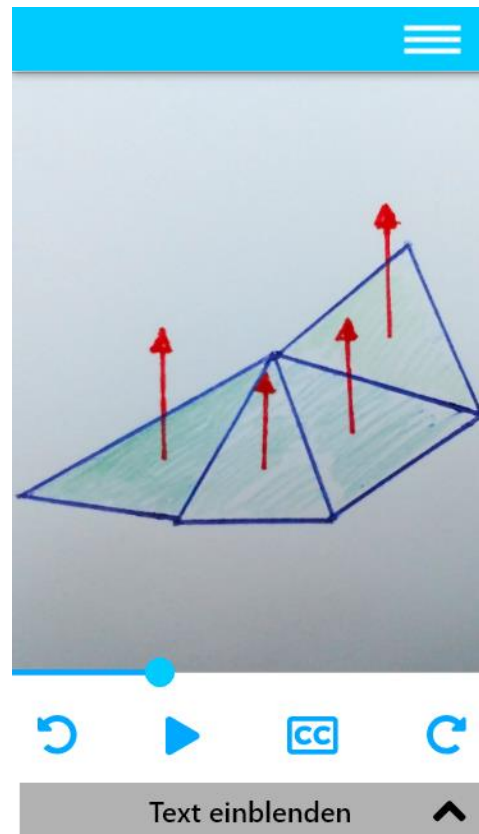
Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>100601</b> Flat-Shading, manchmal auch Constant-Shading genannt, ist ein sehr einfaches Schattierungsverfahren, da pro Polygon nur eine Farbe möglich ist. Wenn man z.B. ein Dreieck als Polygon nimmt, wird der Farbwert unter anderem aus dem Flächen-Normalenvektor, der Flächenfarbe und der Lichtintensität berechnet.</p> <p>Danach werden alle Pixel des Polygons auf diese Farbe gesetzt. Als Ergebnis erhält man dann, besonders bei gekrümmten Oberflächen, eine Facettenartige Darstellung. Das ist auch der größte Nachteil des Flat-Shading, weil es dadurch zum sogenannten Mach-Band-Effekt kommt, wodurch die entstandenen Kanten besonders stark vom menschlichen Auge wahrgenommen werden.</p> <p><b>100602</b> Um die facettenartige Darstellung zu vermindern, muss die Anzahl der Polygone erhöht werden, wodurch der Rechenaufwand aber steigt. Aufgrund der genannten Nachteile kommt das Flat-Shading meistens bei Objekten mit ebenen</p>	<p><b>100601</b> - Einfaches Schattierungsverfahren - Eine Farbe pro Polygon - Facettenartige Darstellung führt zu Mach-Band-Effekt</p> <p><b>100602</b> - Anzahl der Polygone erhöhen, um Darstellung zu verbessern</p>	<p><b>100601</b> Anhand eines Polygons wird gezeigt wie die Fläche, mithilfe des Flächen-Normalenvektors, gefärbt wird. Danach sieht man eine Kugel die aus Polygonen besteht.</p> <p><b>100602</b> Die Anzahl der Polygone wird erhöht, um zu zeigen wie sich der Mach-Band-Effekt verringert.</p>

	Flächen wie z.B. Quader, Würfel oder Pyramiden zum Einsatz.		
--	---	--	--

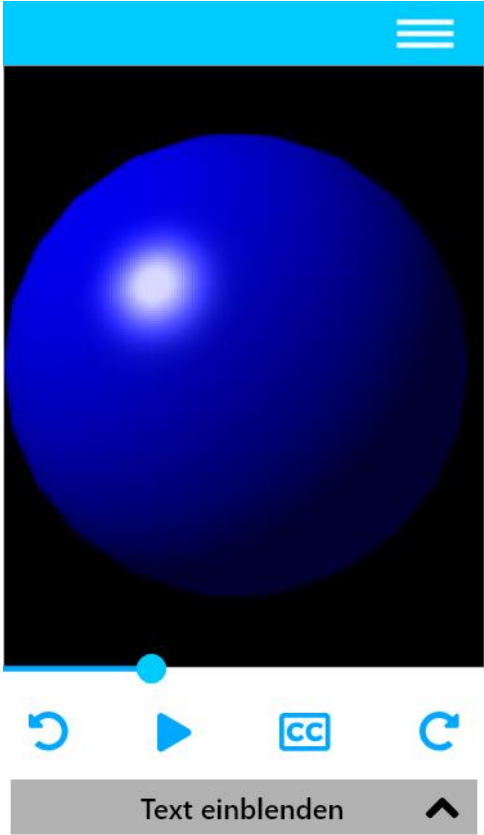


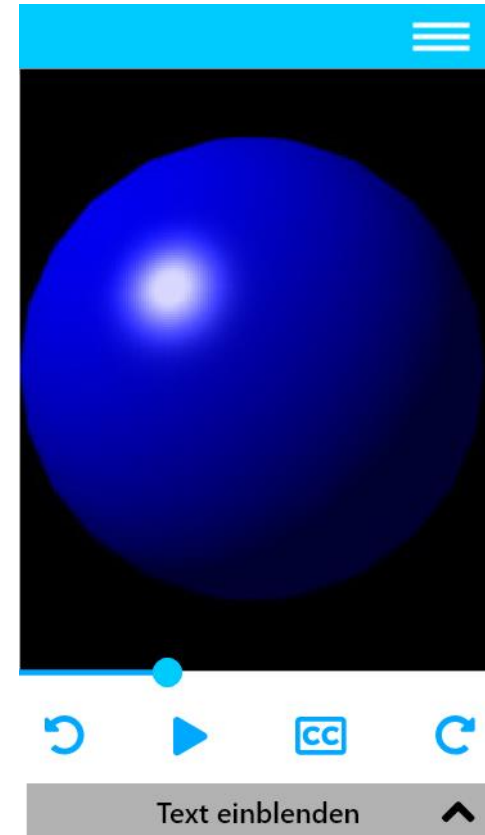
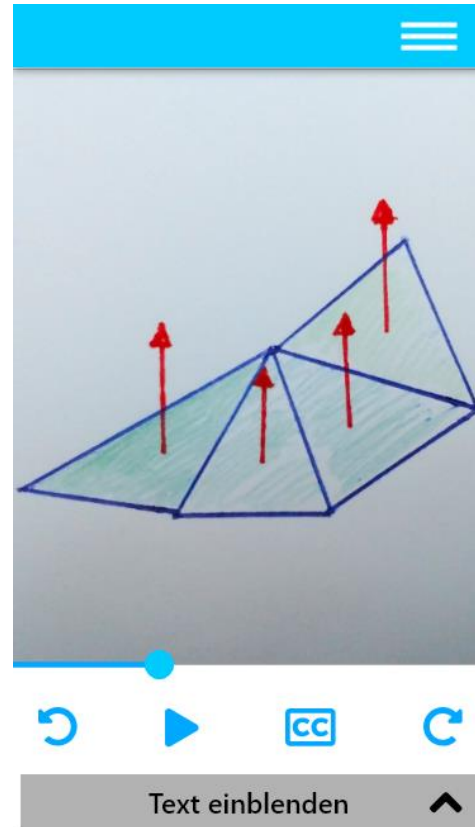
## 10.7 (A) Gouraud-Shading

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>100701</b> Das Gouraud-Shading wurde nach seinem Entwickler Henri Gouraud benannt, der es erstmals 1971 vorstellte.</p> <p><b>100702</b> Das Besondere am Gouraud-Shading ist, dass im Gegensatz zum Flat-Shading, Farbverläufe dargestellt werden können. Dafür werden die Normalenvektoren an den Vertices berechnet. Diese erhält man durch den Mittelwert der Normalen aller angrenzenden Polygone. Danach werden durch Interpolation die Farbwerte an den Vertices berechnet.</p> <p><b>100703</b> Durch dieses Verfahren erscheinen die Kanten der Polygone weniger hart, wodurch Objekte besser rund oder gekrümmt dargestellt werden können. Ein Nachteil ist die bei manchen Objekten fehlerhafte Darstellung von Glanzlichtern und das Vorkommen von Sprüngen im Farbverlauf.</p>	<p><b>100701</b> - 1971 vorgestellt</p> <p><b>100702</b> - Darstellung von Farbverläufen - Berechnung mit Hilfe von Normalenvektoren</p> <p><b>100703</b> - Für gekrümmte Objekte geeignet - Nachteil: Fehlerhafte Darstellung von Glanzlichtern</p>	<p><b>100701</b> Der Erfinder wird kurz eingeblendet.</p> <p><b>100702</b> Anhand eines Polygons wird gezeigt, wie die Fläche gefärbt wird. Dafür werden die Normalenvektoren beispielhaft an einem Polygon dargestellt.</p> <p><b>100703</b> Später sieht man eine Kugel die aus Polygonen besteht.</p>



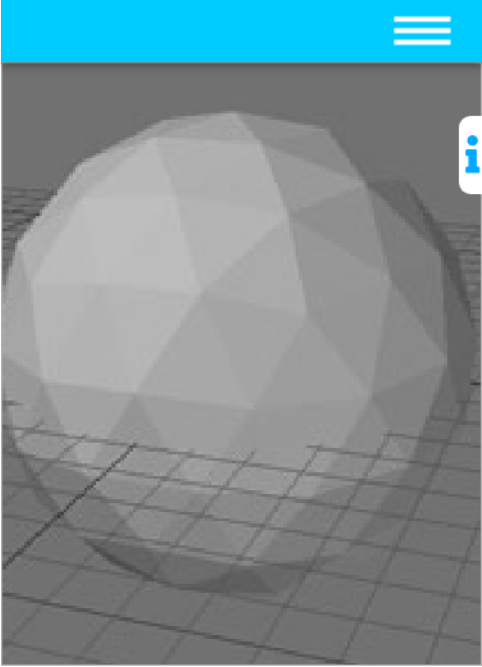
## 10.8 (A) Phong-Shading

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>100801</b> Das Phong-Shading, benannt nach seinem Entwickler Bui Tường Phong, wurde erstmals 1975 vorgestellt.</p> <p><b>100802</b> Bei diesem Verfahren werden zu Beginn, wie beim Gouraud-Shading auch, die Normalen an den Vertices eines Polygons berechnet. Daraufhin wird beim Einfärben der Pixel eine Normale zwischen den Eckpunktnormalen interpoliert, mit der Die Farbe entsprechend der Beleuchtung ausgewertet wird. Dadurch erhält jedes Pixel die entsprechende Beleuchtung und somit auch die korrekte Farbe.</p> <p><b>100803</b> Das Phong Shading liefert aufgrund von glänzenden Oberflächen und Specular highlights, welche hier für den hellen Fleck sorgen, ein realistischeres Ergebnis als das Gouraud Shading, ist aber auch deutlich rechenintensiver.</p>	<p><b>100801</b> - 1975 vorgestellt</p> <p><b>100802</b> - Berechnung mit Normalen - Beleuchtung für jeden einzelnen Pixel</p> <p><b>100803</b> - Nachteil: rechenintensiver</p>	<p><b>100801</b> Der Erfinder wird kurz eingeblendet.</p> <p><b>100802</b> Anhand eines Polygons wird gezeigt wie die Fläche mithilfe der Normalen gefärbt wird.</p> <p><b>100803</b> Danach sieht man eine Kugel die aus Polygonen besteht.</p>

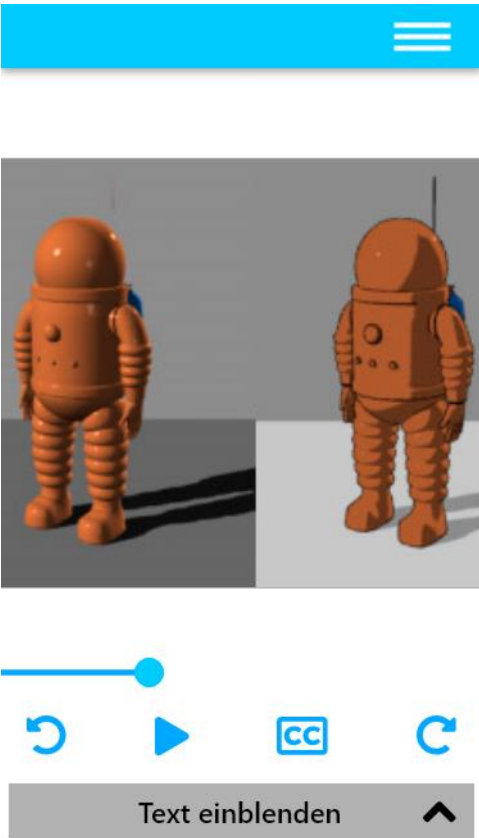




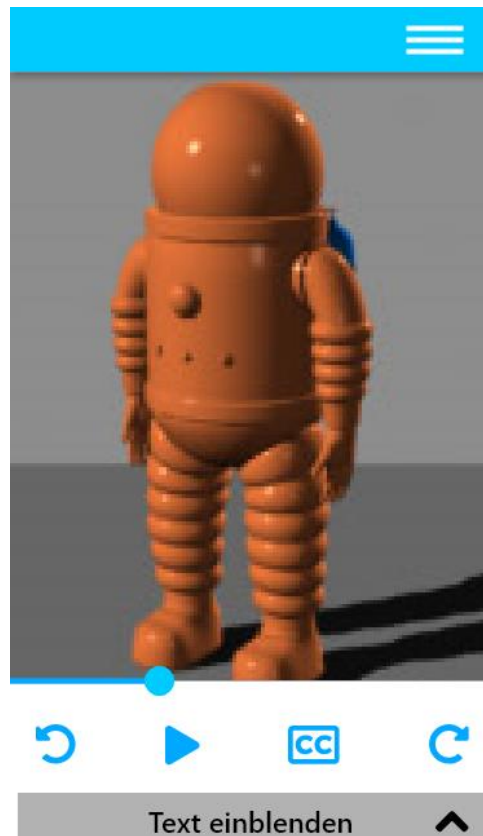
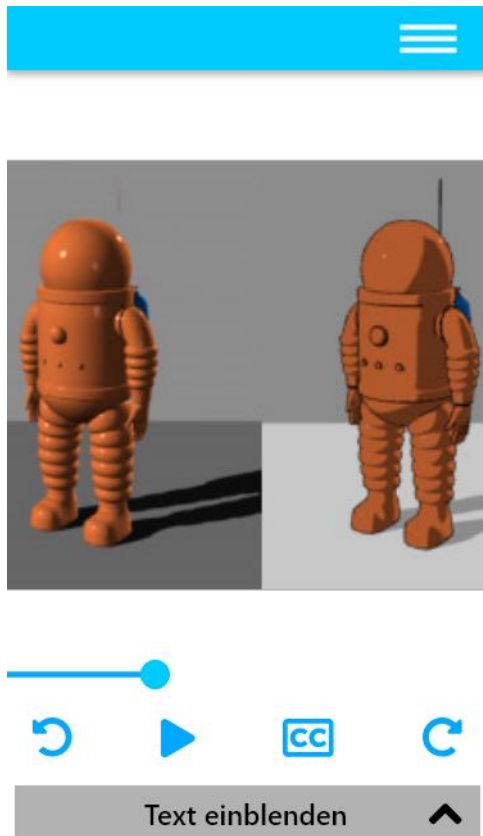
## 10.9 (I) Vergleich zwischen Flat-, Gouraud - und Phong-Shading

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
 <p>Gouraud-Shading <input type="radio"/></p> <p>Flat-Shading <input checked="" type="radio"/></p> <p>Phong-Shading <input type="radio"/></p>	<p><b>100901</b></p> <p>Hier kannst du die verschiedenen Shader anhand eines Objekts vergleichen, indem du unten einen Shader auswählst.</p>		<p><b>100901</b></p> <p>Man kann die verschiedenen Shader an einem Objekt, in dem Fall einer Kugel, anwenden. Durch diese Interaktion kann man gut erkennen, welche Unterschiede es zwischen den Shadern gibt und welchen Einfluss sie auf das Endergebnis haben.</p>

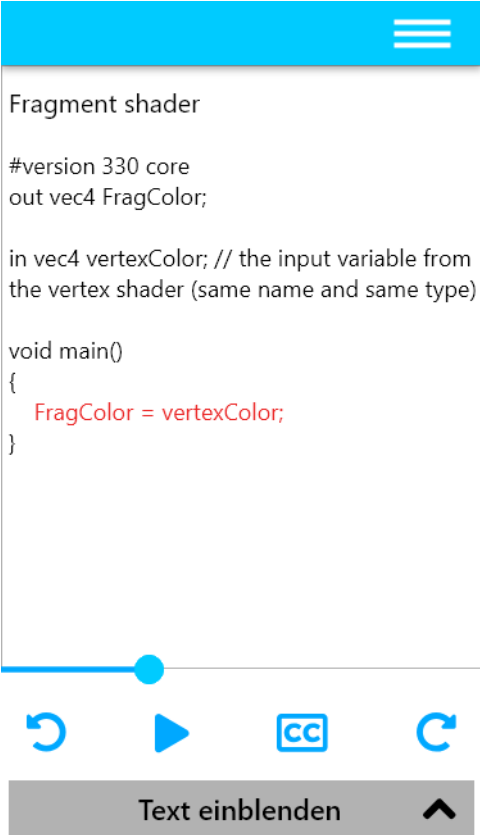
## 10.10 (A) Toon-/Cel-Shading

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>101001</b> Beim Toon-Shading, auch Cel Shading genannt, handelt es sich um eine Technik zum nicht-fotorealistischen Rendern von 3D-Computergrafiken. Als Ergebnis erhält man bei diesem Verfahren eine Optik, die der von gezeichneten Comics oder Zeichentrickfilmen entspricht.</p> <p><b>101002</b> Um diesen Effekt zu erhalten, wird auf weiche Verläufe verzichtet und es kommen nur drei oder vier Helligkeitsstufen zum Einsatz. In der Regel sind das weiß, hellgrau und dunkelgrau. Außerdem verzichtet man meistens auf eine Textur und verwendet nur einzelne Farbtöne.</p> <p><b>101003</b> Um die schwarzen Linien zu erhalten, welche die innere und äußere Kontur des Objekts darstellen, invertiert man Polygone die aufgrund der Perspektive nicht sichtbar wären. Dafür wird das Backface Culling, welches nicht sichtbare Polygone aufgrund der Performanceverbesserung entfernt, rückgängig gemacht. Dies wird teilweise mehrmals mit leichten Variationen durchgeführt,</p>	<p><b>101001</b> - nicht-fotorealistisches Rendern - Comic-Optik</p> <p><b>101002</b> - 3-4 Helligkeitsstufen - einzelne Farbtöne</p> <p><b>101003</b> - Intervenieren von Polygonen um Konturen zu erhalten</p>	<p><b>101001</b> Zuerst wird gezeigt wie ein Toon-Shader in der Praxis aussieht.</p> <p><b>101002</b> Dann wird Schritt für Schritt gezeigt, wie der Toon-Shader auf ein normales Objekt angewendet wird. Als erstes werden die Farben und die Helligkeitsstufen reduziert.</p> <p><b>101003</b> Als nächstes wird das Backface Culling rückgängig gemacht, um eine schwarze Linie als Kontur zu erhalten. Das wird an einem Ausschnitt des Objekts gezeigt, an dem sich eine Krümmung befindet.</p>

	um eine bessere Kontur zu erhalten. Die neu hinzugefügten Polygone werden zum Schluss schwarz gefärbt.		
--	--	--	--



## 10.11 (A) Programmierung eines Shaders

Screen	Sprechertexte	Stichwörter / Notizen	Regieanweisungen
	<p><b>101101</b> In diesem Kapitel bekommst du einen Einblick in den Code eines simplen Shader-Programmes, Der Code eines Shaders erinnert stark an die Programmiersprache C, weshalb es viele Parallelen zu dieser und auch anderen Programmiersprachen gibt. Dieses Beispiel wurde in der OpenGL Shading Language, kurz GLSL, geschrieben.</p> <p><b>101102</b> Shader haben in der Regel folgenden Aufbau. Als erstes wird die Version deklariert, gefolgt von den Input und Outout Variablen. Bei Uniform werden Daten von der CPU zur GPU transportiert. In der Main-Funktion werden die Inputs verarbeitet und das Ergebnis an die Output-Variablen weitergegeben.</p> <p><b>101103</b> In diesem Beispiel wird ein Vertex Shader verarbeitet. Als Input wird ein Vektor mit einer Position benutzt, welcher als Output eine Farbe erhalten soll. Diese wird innerhalb der Main-Funktion zugewiesen.</p>		<p><b>101101-04</b> Die angesprochenen Teile des Codes werden farblich hervorgehoben.</p>

	<p><b>101104</b> Der Outout vom Vertex Shader wird dann als Input beim Fragment Shader genutzt. Dort wird die Farbe für ein Fragment übernommen.</p> <p><b>101105</b> Als Ergebnis erhält man ein Fragment mit der zuvor einprogrammierten Farbe.</p>		<p><b>101105</b> Das farbige Fragment wird eingeblendet.</p>
--	---	--	--

```
#version version_number
in type in_variable_name;
in type in_variable_name;

out type out_variable_name;

uniform type uniform_name;

void main()
{
    // process input(s) and do some weird
    graphics stuff
    ...
    // output processed stuff to output variable
    out_variable_name =
    weird_stuff_we_processed;
}
```

Text einblenden ^

```
Vertex shader

#version 330 core
layout (location = 0) in vec3 aPos; // the
position variable has attribute position 0

out vec4 vertexColor; // specify a color output
to the fragment shader

void main()
{
    gl_Position = vec4(aPos, 1.0); // see how we
    directly give a vec3 to vec4's constructor

    vertexColor = vec4(0.5, 0.0, 0.0, 1.0); // set
    the output variable to a dark-red color
}
```

Text einblenden ^

```
Fragment shader

#version 330 core
out vec4 FragColor;

in vec4 vertexColor; // the input variable from
the vertex shader (same name and same type)

void main()
{
    FragColor = vertexColor;
}
```

Text einblenden ^

