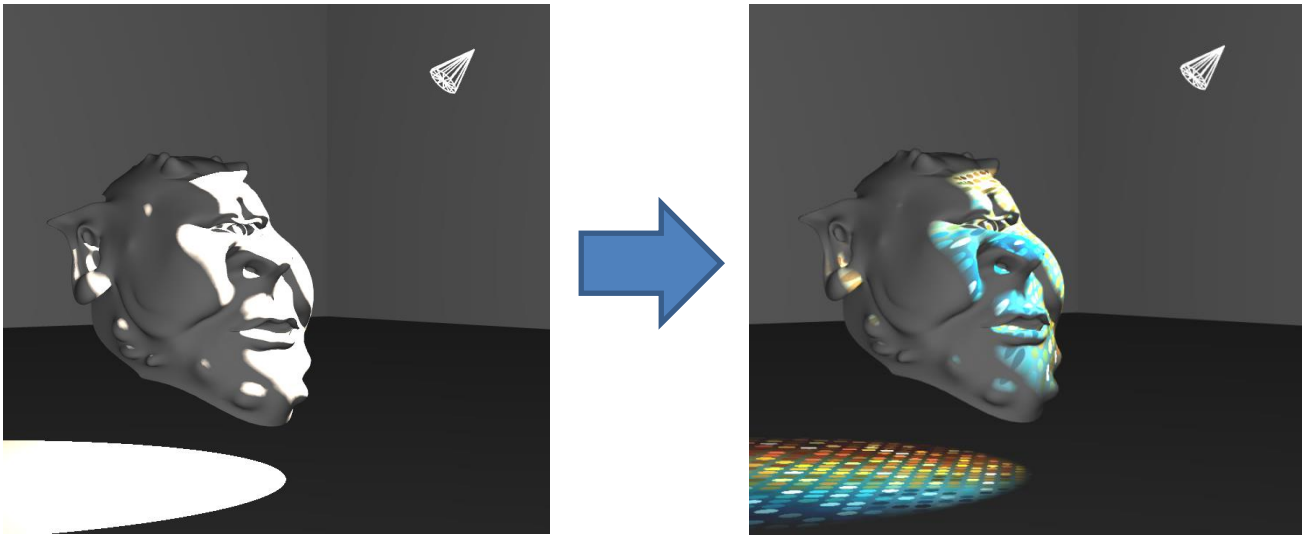
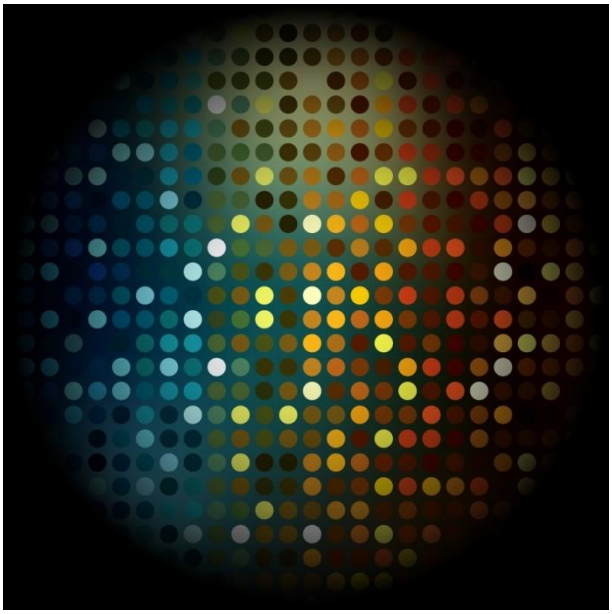
Aufgabenstellung:

Gegeben ist eine einfache Szene, die von einem Spotlight beleuchtet wird. Ihre Aufgabe ist es, dem Spotlight ein sogenanntes Gobo hinzuzufügen, d.h. das einfallende Licht soll nicht mehr konstant sein, sondern je nach Lichteinfallsrichtung variieren. Anders ausgedrückt, das Gobo soll ausgehend von der Lichtquelle in die Szene projiziert werden.



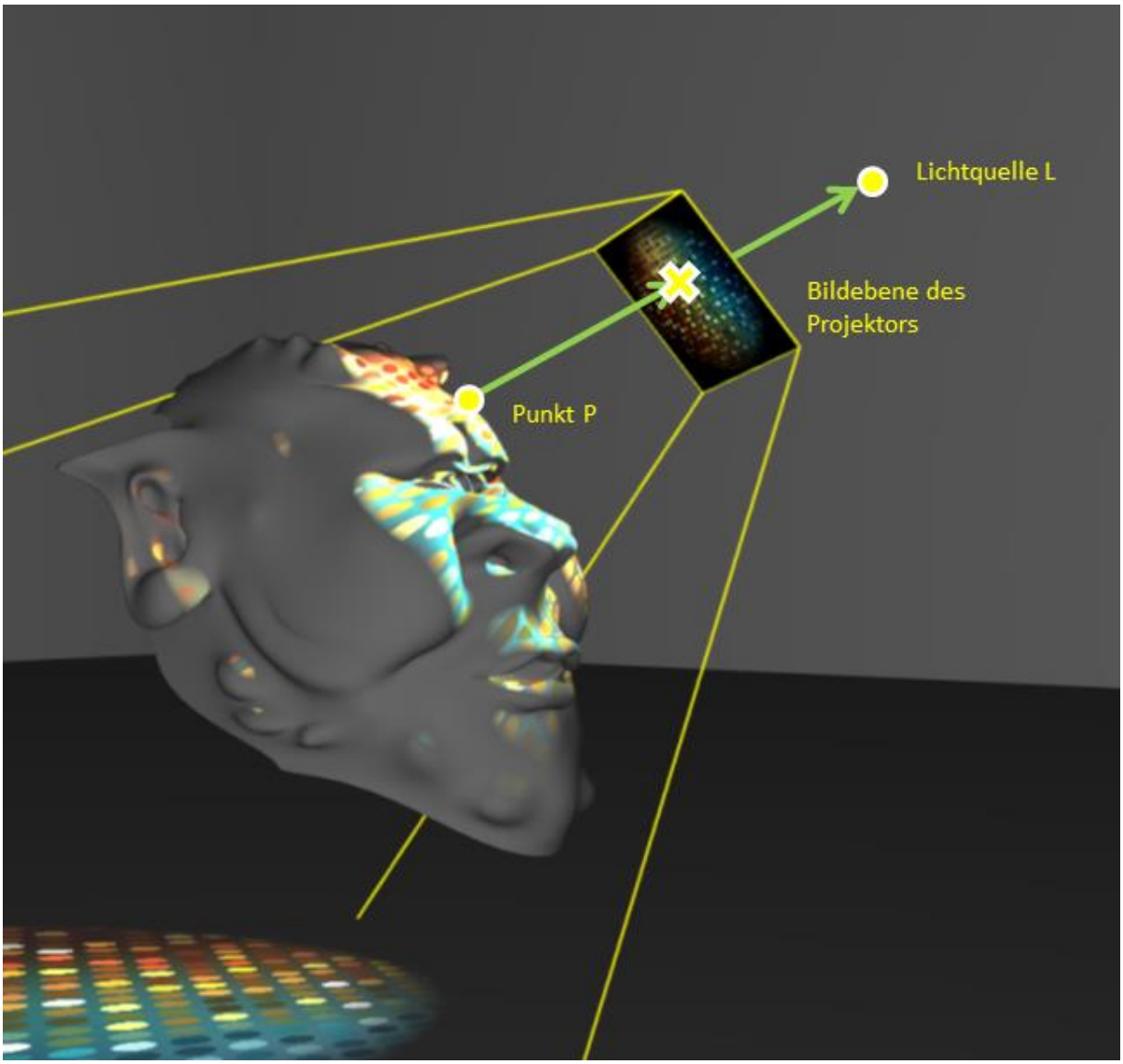
Das Gobo an sich ist nichts weiter als eine 2D Textur:



Das heißt, Sie werden den entsprechenden Farbwert aus der Textur auslesen und in die Beleuchtungsberechnungen einfließen lassen.

Die Challenge hierbei ist das Erzeugen der Texturkoordinaten, denn ganz offensichtlich helfen Ihnen die normalen, von einem Artist erzeugten UV Koordinaten der Objekte hier nicht weiter. **Die Projektion hängt von Position & Orientierung der Modelle bezüglich der projizierenden Lichtquelle ab.**

Betrachten Sie folgende Skizze, in der sich Lichtquelle / Projektor und Geometrie befinden



Wir möchten die Lichtfarbe für einen Punkt P in der Szene bestimmen -- P wird natürlich ein Fragment sein.

Eine Möglichkeit wäre es, einen von P zur Lichtquelle L einen Strahl aufzustellen, und diesen mit der Bildebene zu schneiden: Über den Schnittpunkt sind die Texturkoordinaten bestimmbar.

In der Computergrafik bevorzugen wir allerdings einen anderen Ansatz. Was Ihnen beim Betrachten der Grafik eigentlich auffallen sollte, ist, dass Sie ein **perspektivisches Kamerafrustum** vor sich haben:

Die Kameraposition entspricht der Light Position, die Blickrichtung der Spot Direction. Die in der Grafik eingezeichnete Bildebene ist die Near Plane der Kamera. Sie können sich quasi vorstellen dass die Gobo Textur direkt am Viewport der Lichtkamera aufgetragen ist: **Jedes Pixel im Viewport der Lichtkamera entspricht genau einem Texel der 2D Textur.**

Wenn Sie also den Punkt P in den Viewport -- bzw. in den Screen Space -- dieser Lichtkamera transformieren, dann bekommen Sie die Texturkoordinaten. Auf diese Weise nutzen Sie Matrizenmultiplikationen, was auf der GPU weit performanter zu berechnen ist als ein Strahl / Ebenenschnittpunkt. Zur Erinnerung:

- mit **modelTf** multiplizieren -- World Space
- mit **viewprojectionTf** der Lichtkamera multiplizieren -- Clip Space der Lichtkamera
- perspektivische Division -- Normalized Device Space der Lichtkamera
- mit **viewportTf** der Lichtkamera multiplizieren -- x/y wären dann die texelFetch Lookup Koordinaten

D.h. Sie brauchen die View- & Projektions-, und Viewport Matrix der Lichtkamera.

- View Matrix: wird durch Light Position / Spot Direction eindeutig bestimmt. (Center of interest = light position + spot direction, und die up direction darf halt nicht gerade parallel zur spot direction sein)
- Projection Matrix: das vertikale FoV entspricht dem Spot Angle * 2, die Aspect Ratio passt sich an die 2D Textur an. Für die Near & Far Planes werden Werte gewählt, die sicherstellen dass alle beleuchtete Geometrie innerhalb des Kamerafrustums liegt.
- Viewport Matrix: der Viewport beginnt bei (0,0) und hat die Dimensionen der Textur.

Allerdings brauchen Sie die Viewport Matrix letztlich doch nicht: nach der perspektivischen Division sind die Koordinaten im Normalized Device Space, d.h. die x/y/z Komponenten sind im Wertebereich [-1, 1]. Wenn Sie diese nach [0, 1] transformieren, haben Sie verwendbare UV Koordinaten und können mit dem Befehl texture die Textur auslesen – incl. hardwareunterstützter Interpolation, was bei diesem Anwendungsfall sowieso erwünscht ist.

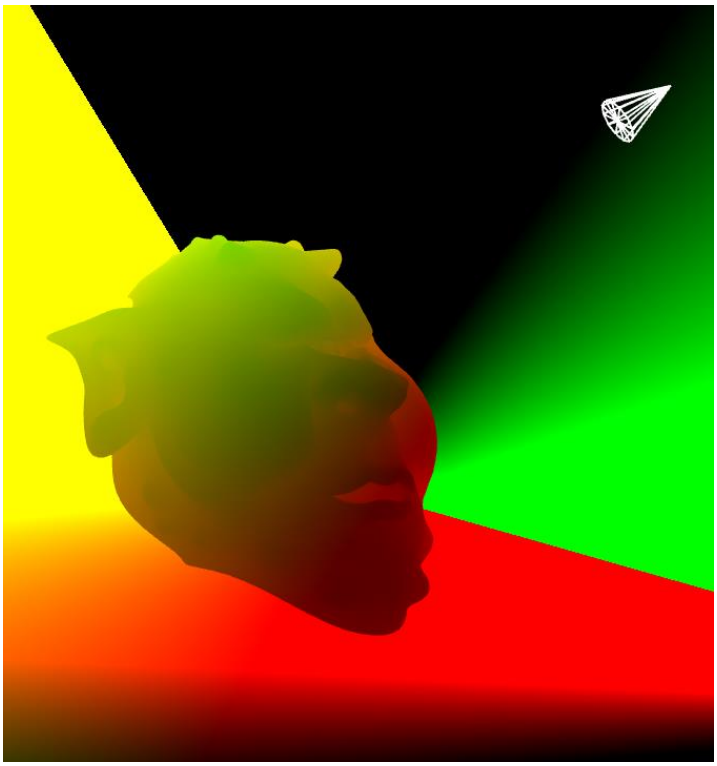
Schritt 1

Stellen Sie mit glm::lookAt / glm::perspective die beiden Kameramatrizen auf, und übergeben Sie diese an das GLSL Programm.

Berechnen Sie im Vertex Shader die Vertexposition im Camera Space der Lichtquelle (im Folgenden **lightSpacePosition** genannt), und geben Sie diesen Wert als Output an den Fragment Shader weiter.

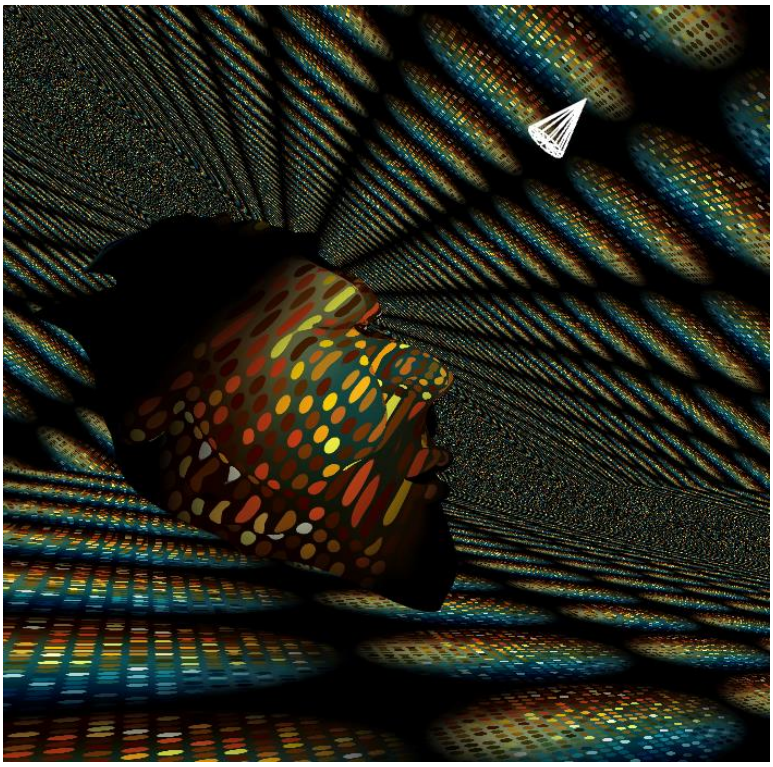
Nur damit das klar ist: gl_Position müssen Sie nach wie vor mit den normalen Kameramatrizen der Renderkamera multiplizieren, Sie möchten die Szene ja aus der Sicht der Renderkamera darstellen. Es gibt jetzt also zwei Kameras im Shader, lassen Sie sich davon nicht verwirren.

Im Fragment Shader führen Sie die perspektivische Division der **lightSpacePosition** durch, und transformieren die XY-Koordinaten in den Wertebereich [0, 1] - dies sind Ihre UV Koordinaten. Danach geben Sie, zu Testzwecken, die UV Koordinaten als Farbwerte am Screen aus -> sollte wie Folgt aussehen:



Schritt 2

Binden und übergeben Sie die Gobo Texture an den Fragment Shader. Lesen Sie im Fragment Shader mit den zuvor berechneten UV Koordinaten die Textur aus, und geben Sie nun diesen Wert aus:



Übrigens hängt der Output auch vom Texture Wrap Modus der Textur ab – in diesem Beispiel auf GL_REPEAT eingestellt; Bei GL_CLAMP / GL_CLAMP_TO_BORDER wäre das Ergebnis weniger freakig.

Schritt 3

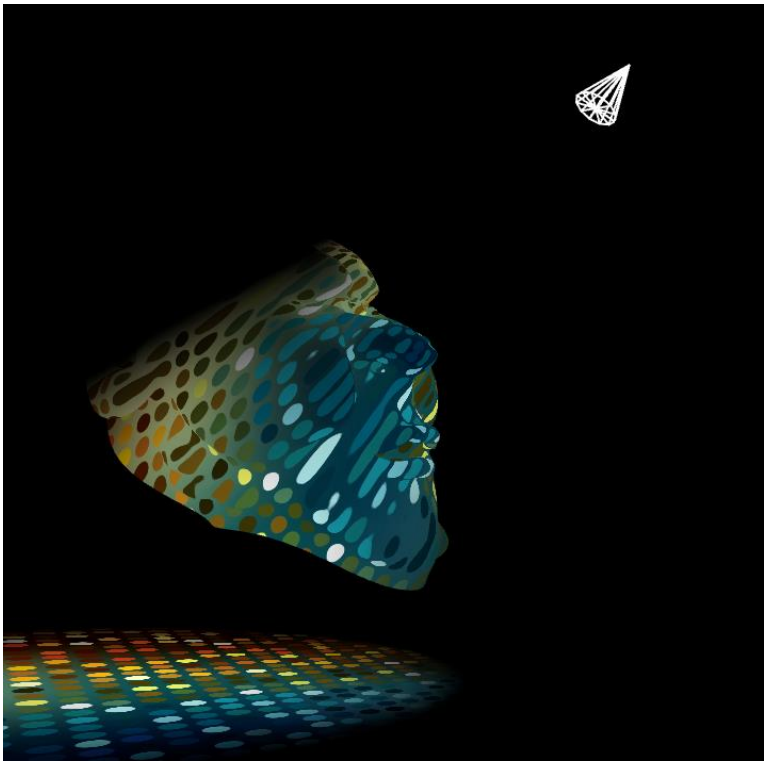
Was Ihnen noch fehlt, ist das Clipping gegen das Kamerafrustum. Sie können einfach die Normalized Device Koordinaten abtesten, es müssen die X/Y/Z Komponenten im Wertebereich [-1, 1] liegen! Falls dies nicht zutrifft, liegt das Fragment außerhalb des Kamerafrustums und wird nicht beleuchtet.

Wichtig: bitte lesen Sie die Gobo Textur jedenfalls aus, auch wenn das Fragment außerhalb des Frustums liegt, d.h. machen Sie nicht so etwas:

```
if ( in frustum ) {
	fragmentColor = texture(goboTex, uv);
} else {
	fragmentColor = …;
}
```

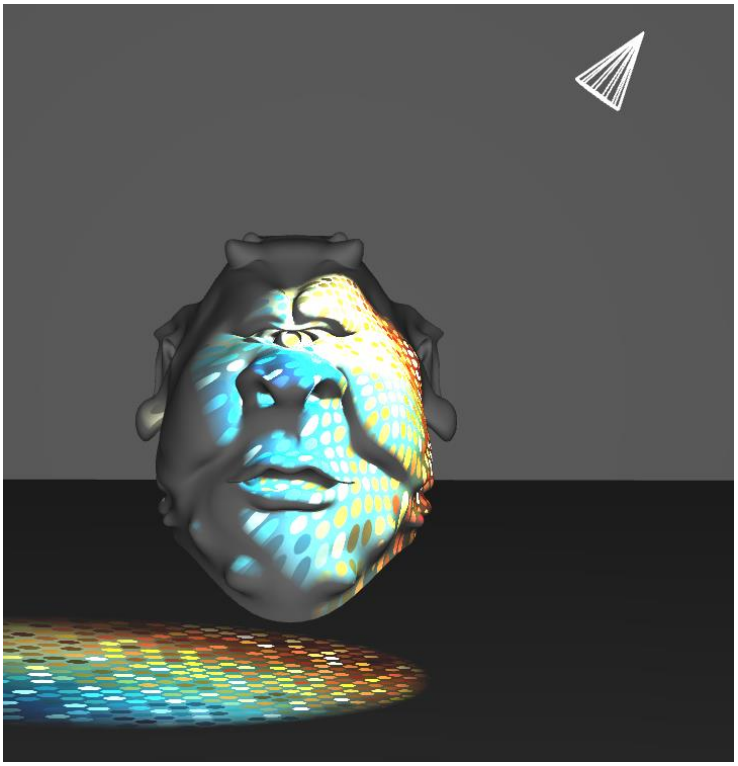
Die Grafikkarte verträgt es nicht, wenn man einen bedingten, interpolierenden -- also mit texture, texelFetch ist ok! -- Zugriff in eine Textur macht, sofern die Bedingung nicht an einer Uniform Variablen hängt, sondern vom Fragment gesteuert wird ist. (undefined Behaviour, mehr Infos gibt hier)

Lesen Sie also den Wert aus der Textur jedenfalls aus, und falls das Fragment nicht im Frustum liegt, verwenden Sie ihn einfach nicht ☺



Schritt 4 – Ende

Anstatt einfach nur die Texturfarbe auszugeben, multiplizieren Sie die Farbe mit dem Beitrag des Spotlights & geben die Beleuchtung aus.



Weitere Anmerkungen: Da die hier vorgestellte Art des Texturzugriffs häufig benutzt wird, gibt es dafür einen eigenen GLSL Befehl – textureProj. Wir verwenden diesen Befehl in der LVA nicht, da man ihn eh nicht sinnvoll nutzen kann wenn man nicht weiß, was eigentlich passiert; er wird jedoch gerne in Online Tutorials eingesetzt, besonders zum Thema Shadow Mapping. Bei den Hausübungen sollen ist der Befehl verboten ☺