

Übungen - Bildgenierung

Übung 07.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 13, 2023



Table of Contents

- 1 Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong
- 2 Aufgabe 19: Rundflug um den Eiffelturm
- 3 Aufgabe 20: Perspektivische Projektion mit OpenGL
- 4 Aufgabe 20: z-Buffer mit OpenGL



Table of Contents

1 Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

2 Aufgabe 19: Rundflug um den Eiffelturm

3 Aufgabe 20: Perspektivische Projektion mit OpenGL

4 Aufgabe 20: z-Buffer mit OpenGL



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

Implementieren Sie Hierzu in der Funktion

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

das Beleuchtungsmodell nach Phong für *eine* Lichtquelle



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

VecRGB: Für Lichter stellen diese Werte die Lichtintensität des jeweiligen Farbkanals dar, für Materialien bzw. Objektoberflächen deren Reflexionskoeffizienten für den jeweiligen Farbkanal.



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

```
Vec3D ecke //Position der Ecke in Weltkoordinaten
Vec3D normale //Normale zur Fläche in dieser Ecke (in
→ Weltkoordinaten)
Vec3D Auge //Koordinaten des Auges in Weltkoordinaten
Vec3D licht //Koordinaten der Lichtquelle in Weltkoordinaten
VecRGB lightAmbient //ambiente Lichtintensität
VecRGB lightDiffuse //diffuse Lichtintensität
VecRGB lightSpecular //Lichtintensität für winkelabhängige Reflexion
VecRGB materialAmbient //ambienter Reflexionskoeffizient des
→ Materials
VecRGB materialDiffuse //diffuser Reflexionskoeffizient des
→ Materials
VecRGB materialSpecular //winkelabhängiger Reflexionskoeffizient des
→ Materials
double materialSpecularity //Exponent für winkelabhängige Reflexion
double c0, c1, c2 //Konstanten für entfernungsabhängige Dämpfung
```

Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Um nicht jeden Farbkanal einzeln berechnen zu müssen sind neben + und - geeignete Operatoren vorgegeben, z. B. VecRGB v1 * VecRGB v2 für elementweise Multiplikation.



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Um nicht jeden Farbkanal einzeln berechnen zu müssen sind neben $+$ und $-$ geeignete Operatoren vorgegeben, z. B. $\text{VecRGB } v1 * \text{VecRGB } v2$ für elementweise Multiplikation.

Im Falle eines negativen Skalarproduktes ist der entsprechende Lichtanteil auf Null zu setzen, um keine negativen Lichtintensitäten zu erzeugen.



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Es gibt 2 Richtungen, die sehr wichtig sind:

- 1 Blickrichtung v .
- 2 Lichtrichtung l .

Wie können wir die rechnen?



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Beleuchtungsmodell.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Es gibt 2 Richtungen, die sehr wichtig sind:

- 1 Blickrichtung v .
- 2 Lichtrichtung l .

Wie können wir die rechnen?

- 1 $v = \text{augen} - \text{ecke}$.
- 2 $l = \text{licht} - \text{ecke}$.

Die sind **Richtungen**, d.h., die müssen normalisiert Werden!



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Ambienteslicht.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit dem Ambientes Licht ?



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Ambienteslicht.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit dem Ambientes Licht ?

In diesem Fall sind die angegebenen Parameter wichtig:

```
VecRGB lightAmbient //ambiente Lichtintensität  
VecRGB materialAmbient //ambienter Reflexionskoeffizient des  
↪ Materials
```



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Ambienteslicht.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit dem Ambientes Licht ?

In diesem Fall sind die angegebenen Parameter wichtig:

```
VecRGB lightAmbient //ambiente Lichtintensität  
VecRGB materialAmbient //ambienter Reflexionskoeffizient des  
↪ Materials
```

Was ist denn "**VecRGB** ambient" ? (akk intensität)



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Ambienteslicht.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit dem Ambientes Licht ?

In diesem Fall sind die angegebenen Parameter wichtig:

```
VecRGB lightAmbient //ambiente Lichtintensität  
VecRGB materialAmbient //ambienter Reflexionskoeffizient des  
↪ Materials  
/*VecRGB v1 * VecRGB v2 für elementweise Multiplikation.*/
```

Was ist denn **VecRGB** ambient ?

- $I_a = \text{ambient} = \text{lightAmbient} * \text{materialAmbient}$. (7 – 16)



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit der **diffusen Reflexion**?



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit der **diffusen Reflexion**?

In diesem Fall sind die angegebenen Parameter wichtig:

```
Vec3D normale //Normale zur Fläche in dieser Ecke (in  
→ Weltkoordinaten)  
Vec3D l //Lichtrichtung (gerechnet)  
VecRGB lightDiffuse //diffuse Lichtintensität  
VecRGB materialDiffuse //diffuser Reflexionskoeffizient des  
→ Materials
```

Wie berechnet man die Intensität von diffus reflektiertem Licht?



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

Was ist mit der **diffusen Reflexion**?

In diesem Fall sind die angegebenen Parameter wichtig:

```
Vec3D normale //Normale zur Fläche in dieser Ecke (in  
↪ Weltkoordinaten)  
VecR3D l //Lichtrichtung (gerechnet)  
VecRGB lightDiffuse //diffuse Lichtintensität  
VecRGB materialDiffuse //diffuser Reflexionskoeffizient des  
↪ Materials
```

Wie berechnet man die Intensität von diffus reflektiertem Licht? (7-18)

$$I_d = \langle \text{normale}, l \rangle * \text{lightDiffuse} * \text{materialDiffuse}$$



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

Was ist mit der **diffusen Reflexion**?

In diesem Fall sind die angegebenen Parameter wichtig:

```
Vec3D normale //Normale zur Fläche in dieser Ecke (in  
↪ Weltkoordinaten)  
VecR3D l //Lichtrichtung (gerechnet)  
VecRGB lightDiffuse //diffuse Lichtintensität  
VecRGB materialDiffuse //diffuser Reflexionskoeffizient des  
↪ Materials
```

Wie berechnet man die Intensität von diffus reflektiertem Licht? (7-18)

$$I_d = \langle \text{normale}, l \rangle * \text{lightDiffuse} * \text{materialDiffuse}$$

Fast: "Im Falle eines negativen Skalarproduktes ist der entsprechende Lichtanteil auf Null zu setzen, um keine negativen Lichtintensitäten zu erzeugen."



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

Wie berechnet man die Intensität von diffus reflektiertem Licht? (7-18)

$$I_d = \langle \text{normale}, l \rangle * \text{lightDiffuse} * \text{materialDiffuse}$$

```
double dotNL = skalarprod(normale, l1);  
double lambert = max(0.0, dotNL); // immer größer als 0  
VecRGB diffuse = lambert * lightDiffuse * materialDiffuse;
```



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

winkelabhängige Reflexion

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit der **winkelabhängige Reflexion**? (7-19)



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

winkelabhängige Reflexion

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit der **winkelabhängige Reflexion**? (7-19)

Wie können wir s rechnen?

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l;
```



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

winkelabhängige Reflexion

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit der **winkelabhängige Reflexion**? (7-19)

Wie können wir s rechnen?

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l;
```

$$I_w = (v^T \cdot s)^{v_k} * I_l * R$$

Wir kennen schon s , und die andere Variablen?



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

winkelabhängige Reflexion

Was ist mit der **winkelabhängige Reflexion**? (7-19)

Wie können wir s rechnen?

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l;
```

$$I_w = (v^T \cdot s)^{v_k} * I_l * R$$

Wir kennen schon s , und die andere Variablen?

```
double materialSpecularity //Exponent für winkelabhängige Reflexion
↪  $v_k$ 
VecRGB lightSpecular //Lichtintensität für winkelabhängige Reflexion
↪  $I_l$ 
VecRGB materialSpecular //winkelabhäng. Reflexionskoeff. des
↪ Materials  $R$ 
Vec3D v = auge - ecke // Blickrichtung  $v$ 
```



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

winkelabhängige Reflexion

$$I_w = (v^T \cdot s)^{v_k} * I_l * R$$

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,
const Vec3D& auge, const Vec3D& licht,
const DrawColour& farbe){
    ...
    ...
    // winkelabhängige Reflexion
    Vec3D s1 = s / norm(s);
    double cospsi = max(0.0, skalarprod(v1, s1));
    double cospsiny = pow(cospsi, materialSpecularity);
    VecRGB specular = cospsiny * lightSpecular * materialSpecular;
}
```



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Entfernungsabhängige Dämpfung

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,  
const Vec3D& auge, const Vec3D& licht,  
const DrawColour& farbe)
```

Was ist mit der **Entfernungsabhängige Dämpfung**? (7-22)

Einfach, wir haben alle c -Werte.



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Entfernungsabhängige Dämpfung

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,
const Vec3D& auge, const Vec3D& licht,
const DrawColour& farbe){
    ...
    ...
    // Entfernungsabhängige Dämpfung
    double att = 1.0 / (c0 + c1 * dl + c2 * dl * dl);
    double attenuation = min(1.0, att);
}
```



Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Entfernungsabhängige Dämpfung

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,
const Vec3D& auge, const Vec3D& licht,
const DrawColour& farbe){
    ...
    ...
    // Entfernungsabhängige Dämpfung
    double att = 1.0 / (c0 + c1 * dl + c2 * dl * dl);
    double attenuation = min(1.0, att);

    // Fertige Farbe
    VecRGB total = ambient + attenuation * (diffuse + specular);
}
```



Table of Contents

- 1 Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong
- 2 Aufgabe 19: Rundflug um den Eiffelturm
- 3 Aufgabe 20: Perspektivische Projektion mit OpenGL
- 4 Aufgabe 20: z-Buffer mit OpenGL



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Schauen wir uns zuerst das Rahmenprogramm an.



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

```
int maindraw()  
{  
    int npics = 101;  
}
```



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

```
int maindraw()
{
    vector<Dreieck> dreiecke;           // die Dreiecke selbst
    ClipQuad clip = ClipQuadDefault;    // clipping Information
    int i;
    Vec3D cop;                          // center of projection = Augenposition
    Vec3D tgt;                          // target = Betrachteter Punkt
    Vec3D vup(0, 1, 0);                // view-up vector = Aufwärtsrichtung
    Matrix4x4 nzen;                    // Transformation zur Normalisierung auf
    ↪ kanon. Bildraum
    int npics = 101;
}
```



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

```
int maindraw()
{
    vector<Dreieck> dreiecke;           // die Dreiecke selbst
    ClipQuad clip = ClipQuadDefault;    // clipping Information
    int i;
    Vec3D cop;                          // center of projection = Augenposition
    Vec3D tgt;                          // target = Betrachteter Punkt
    Vec3D vup(0, 1, 0);                 // view-up vector = Aufwärtsrichtung
    Matrix4x4 nzen;                     // Transformation zur Normalisierung auf
    ↪ kanon. Bildraum
    int npics = 101;
}
```

Und wir lesen das Modell wie folgt:

```
modellEinlesen(dreiecke, cop, tgt);
```


Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

```
int maindraw()
{
    vector<Dreieck> dreiecke;           // die Dreiecke selbst
    ClipQuad clip = ClipQuadDefault;   // clipping Information
    int i;
    Vec3D cop;                          // center of projection = Augenposition
    Vec3D tgt;                          // target = Betrachteter Punkt
    Vec3D vup(0, 1, 0);                // view-up vector = Aufwärtsrichtung
    Matrix4x4 nzen;                     // Transformation zur Normalisierung
                                      // auf kanon. Bildraum

    int npics = 101;

    modellEinlesen(dreiecke, cop, tgt);
}
```

Nichts Neues.

Aber dieses Mal brauchen wir viele (100) vrps und cops.



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Nichts Neues.

Aber dieses Mal werden wir viele (100) vrps und cops brauchen.

```
int maindraw()
{
    int npics = 101;
    /*...
     cop;    tgt;    vup;
    ...*/
    vector<Vec3D> cops(npics);
    vector<Vec3D> vrps(npics);
}
```

Wie groß wird unser Schritt sein? (y-Achse)



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Wie groß wird unser Schritt sein?

```
int maindraw()
{
    int npics = 101;
    .
    .
    vector<Vec3D> cops(npics);
    vector<Vec3D> vrps(npics);

    double step = (550.0 + 662.0) / (npics - 1.0);
    for (i = 0; i < npics; ++i)
        {...}
}
```

Wie lauten die Koordinaten von cops und crps, wenn wir einmal um den Turm herumgehen?



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Wie lauten die Koordinaten von cops und crps, wenn wir einmal um den Turm herumgehen?



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Wie lauten die Koordinaten von `cops` und `crps`, wenn wir einmal um den Turm herumgehen?

Übungsblatt: Halten Sie einen konstanten Abstand von 800 Längeneinheiten von der Mitte des Turms (y -Achse) und blicken Sie immer waagrecht zur Mitte.



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Wie lauten die Koordinaten von `cops` und `crps`, wenn wir einmal um den Turm herumgehen?

Übungsblatt: Halten Sie einen konstanten Abstand von 800 Längeneinheiten von der Mitte des Turms (y -Achse) und blicken Sie immer waagerecht zur Mitte.

i.e. Wie lauten die Koordinaten von `cops` und `crps`, wenn wir einmal um den Turm herumgehen?? (Taffel)



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Wie lauten die Koordinaten von cops und crps, wenn wir einmal um den Turm herumgehen?

```
int maindraw(){
    int npics = 101;
    double step = (550.0 + 662.0) / (npics - 1.0);
    for (i = 0; i < npics; ++i){
        // 3.6° pro Schritt, 360° insgesamt
        cops[i].el[0] = 800 * cos(0.02 * M_PI * i);
        cops[i].el[1] = step * i - 662;
        cops[i].el[2] = ???
        vrps[i].el[0] = ???
        vrps[i].el[1] = ???
        vrps[i].el[2] = ???
    }
```



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Gut! Jetzt, mahlen! 100 Bilder! Wir brauchen auch ein vector von 100-pics und eine for-Schleife...



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Gut! Jetzt, mahlen! 100 Bilder! Wir brauchen auch ein vector von 100-pics und eine for-Schleife...

```
int maindraw(){  
    .  
    .  
    vector<Drawing> pics(npics);  
    Drawing pic(250, 400, 255);  
    pic.show();  
    for (i = 0; i < npics; ++i)  
    {  
        ...  
    }
```



Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Wir legen die Position der Lichtquelle fest. Dann wenden wir unsere Transformationsmatrix an:

```
vector<Drawing> pics(npics);
Drawing pic(250, 400, 255);
pic.show();
for (i = 0; i < npics; ++i)
{
    pic = 255;
    // Position der Lichtquelle
    Vec3D l = standardLichtQuelle(vrps[i], cops[i], vup);

    nzen = berechneTransformation(cops[i], vrps[i], vup, clip,
                                pic.getWidth(), pic.getHeight());
}
```

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zusammen.

Mahlen! Mit unsere *maleDreiecke* Funktion und speichern:

```
vector<Drawing> pics(npics);
Drawing pic(250, 400, 255);
pic.show();
for (i = 0; i < npics; ++i)
{
    pic = 255;
    Vec3D l = standardLichtQuelle(vrps[i], cops[i], vup);
    nzen = berechneTransformation(cops[i], vrps[i], vup, clip,
                                pic.getWidth(), pic.getHeight());

    maleDreiecke(pic, dreiecke, nzen, clip, doClip, cop, l, false,
↪  false);
    //false/true
    pics[i] = pic;
}
```

Table of Contents

- 1 Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong
- 2 Aufgabe 19: Rundflug um den Eiffelturm
- 3 Aufgabe 20: Perspektivische Projektion mit OpenGL
- 4 Aufgabe 20: z-Buffer mit OpenGL



Aufgabe 20: Perspektivische Projektion mit OpenGL

Übungsblatt: Die Projektionsmatrix ist im Rahmenprogramm bereits mittels `glFrustum` gegeben und Ihre Aufgabe besteht zunächst darin, die ModelView-Matrix korrekt zu setzen. Verwenden Sie hierfür den Befehl **`gluLookAt`** der OpenGL Utility Library,

<https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml>



Aufgabe 20: Perspektivische Projektion mit OpenGL

Die Projektionsmatrix ist im Rahmenprogramm bereits mittels `glFrustum` gegeben und Ihre Aufgabe besteht zunächst darin, die `ModelView`-Matrix korrekt zu setzen. Verwenden Sie hierfür den Befehl **`gluLookAt`** der OpenGL Utility Library,

<https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml>

- Eye point?
- Reference point?
- up vector?



Aufgabe 20: Perspektivische Projektion mit OpenGL

Die Projektionsmatrix ist im Rahmenprogramm bereits mittels `glFrustum` gegeben und Ihre Aufgabe besteht zunächst darin, die `ModelView`-Matrix korrekt zu setzen. Verwenden Sie hierfür den Befehl **`gluLookAt`** der OpenGL Utility Library,

<https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml>

- Eye point \rightarrow cop
- Reference point \rightarrow vrp
- up vector? \rightarrow vup



Aufgabe 20: Perspektivische Projektion mit OpenGL

Die Projektionsmatrix ist im Rahmenprogramm bereits mittels `glFrustum` gegeben und Ihre Aufgabe besteht zunächst darin, die `ModelView`-Matrix korrekt zu setzen. Verwenden Sie hierfür den Befehl **`gluLookAt`** der OpenGL Utility Library,

<https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml>

```
gluLookAt( cop.el[0], cop.el[1], cop.el[2],  
           vrp.el[0], vrp.el[1], vrp.el[2],  
           vup.el[0], vup.el[1], vup.el[2] );
```



Aufgabe 20: Perspektivische Projektion mit OpenGL

Jetzt, "muss ein künstliches Clipping auf die durch `clip.uminf`, `clip.umaxf`, `clip.vminf` und `clip.vmaxf` gegebene Ausdehnung auf der Projektionsebene erfolgen."



Aufgabe 20: Perspektivische Projektion mit OpenGL

Jetzt, "muss ein künstliches Clipping auf die durch `clip.uminf`, `clip.umaxf`, `clip.vminf` und `clip.vmaxf` gegebene Ausdehnung auf der Projektionsebene erfolgen."

Ist die **Projektionsebene** gegeben durch $[umin, umax][vmin, vmax]$, ergibt sich der **geclippte Bereich** als

$$[umin \cdot clip.uminf, umax \cdot clip.umaxf] \times [vmin \cdot clip.vminf, vmax \cdot clip.vmaxf] \quad (1)$$

Übungsblatt: Korrektes Clipping kann auf **zwei** Arten erzielt werden:



Aufgabe 20: Perspektivische Projektion mit OpenGL

2. Das Sichtfenster (**Viewport**) kann entsprechend beschränkt werden....

<https://registry.khronos.org/OpenGL-Refpages/gl4/html/glViewport.xhtml>

(+Taffel).



Aufgabe 20: Perspektivische Projektion mit OpenGL

2. Das Sichtfenster (**Viewport**) kann entsprechend beschränkt werden....

<https://registry.khronos.org/OpenGL-Refpages/gl4/html/glViewport.xhtml>

(+Taffel).

```
glViewport(floor( 0.5 * windowSize[0] * (1.0 - clip.uminf) ),  
           floor( 0.5 * windowSize[1] * (1.0 - clip.vminf) ),  
           ceil( windowSize[0] * 0.5 * (clip.uminf + clip.umaxf) ),  
           ceil( windowSize[1] * 0.5 * (clip.vminf + clip.vmaxf) ));
```



Aufgabe 20: Perspektivische Projektion mit OpenGL

2. Das Sichtfenster (**Viewport**) kann entsprechend beschränkt werden.
(**Done!**)



Aufgabe 20: Perspektivische Projektion mit OpenGL

2. Das Sichtfenster (**Viewport**) kann entsprechend beschränkt werden.
(**Done!**)

Anschließend muss die Berechnung der Projektionsmatrix (**glFrustum**) angepasst werden, so dass die Projektion nun für den geclippten Bereich stattfindet.



Aufgabe 20: Perspektivische Projektion mit OpenGL

2. Das Sichtfenster (**Viewport**) kann entsprechend beschränkt werden. (**Done!**)

Anschließend muss die Berechnung der Projektionsmatrix (**glFrustum**) angepasst werden, so dass die Projektion nun für den geclippten Bereich stattfindet.

<https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glFrustum.xml>

```
// Frustum (Pyramidenstumpf des Bildraums) berechnen und setzen  
glFrustum(umin * clip.uminf, umax * clip.umaxf,  
          vmin * clip.vminf, vmax * clip.vmaxf,  
          znear, zfar);
```



Table of Contents

- 1 Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong
- 2 Aufgabe 19: Rundflug um den Eiffelturm
- 3 Aufgabe 20: Perspektivische Projektion mit OpenGL
- 4 Aufgabe 20: z-Buffer mit OpenGL



Aufgabe 21: z-Buffer mit OpenGL

Aktivieren Sie die Verwendung des z-Buffer-Verfahrens mittels der Befehle **glEnable** und **glDepthFunc** mit passenden Parametern.

<https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glEnable.xml>

<https://registry.khronos.org/OpenGL-Refpages/gl4/html/glDepthFunc.xhtml>

Was sind die passenden Parametern?



Aufgabe 21: z-Buffer mit OpenGL

Aktivieren Sie die Verwendung des z-Buffer-Verfahrens mittels der Befehle **glEnable** und **glDepthFunc** mit passenden Parametern.

Was sind die passenden Parametern?

```
...
glEnable(GL_DEPTH_TEST);
//-----> Puffer Initialisieren!
glDepthFunc(GL_LEQUAL);
...
```



Aufgabe 21: z-Buffer mit OpenGL

Aktivieren Sie die Verwendung des z-Buffer-Verfahrens mittels der Befehle **glEnable** und **glDepthFunc** mit passenden Parametern.

Was sind die passenden Parametern?

```
...  
glEnable(GL_DEPTH_TEST);  
//-----> Puffer Initialisieren!  
glDepthFunc(GL_LEQUAL);  
...
```

glClearDepth(1.0): initialisiert den Z-Puffer, indem er komplett mit dem maximalen Tiefenwert von 1.0 gefüllt wird. Dies signalisiert die maximale Entfernung von der Kamera innerhalb des Tiefenpuffers, dessen Bereich von 0,0 (Nähe) bis 1,0 (Ferne) reicht.



