Übungen - Bildgenierung

Übung 07.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 14, 2022

Table of Contents

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 Mater
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. VecRGB v1 * 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 rechnnen?

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 rechnnen?

1. v = augen - ecke.
2. l = licht - ecke.

Die sind **Richtungen**, d.h., die mussen normaliziert 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 = ambient = lightAmbient * 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)
VecR3D 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)
$I_d = <$normale, l $>$ * lightDiffuse * 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 = <normale, l> * lightDiffuse * materialDiffuse
Fast: "Im Falle eines negativen Skalarproduktes ist der entsprechende Lichtanteil auf Null zu setzen, um keine negativen Lichtintensitäten zu erzeugen."
(Wir werden uns bei der Umsetzung darum kümmern.)

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* rechnnen?

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l1;
```

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 rechnnen?

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l1;
```

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

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l1;
```

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

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)
}
```

Das Phong-Beleuchtungsmodell (S. 7-32) verwendet die Intensitäten, die wir gerade berechnet haben. Schauen wir uns den Code an.

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

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

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

Aus früheren Programmen, z.B. proj3, wissen wir, dass wir diese Variablen benötigen:

```
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 zuzammen.

```
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 zuzammen.

```
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 werden wir viele (100) vrps und cops brauchen.

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

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

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

Wie groß wird unser Schritt sein?

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

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 zuzammen.

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] = 800 * sin(0.02 * M_PI * i);
      vrps[i].el[0] = 0;
      vrps[i].el[1] = step * i - 662;
      vrps[i].el[2] = 0;
    }
```

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

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

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] = 800 * sin(0.02 * M_PI * i);
      vrps[i].el[0] = 0;
      vrps[i].el[1] = step * i - 662;
      vrps[i].el[2] = 0;
    }
```

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

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

Gut! Jetz, mahlen? 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 zuzammen.

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

```
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());

  maleDreiecke(pic, dreiecke, nzen, clip, doClip, cop, l, false, false);

  pics[i] = pic;
}
```

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`

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

Jetz, "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

$$[umin \cdot clip.uminf\,, umax \cdot clip.umaxf] \times [vmin \cdot clip.vminf\,, vmax \cdot clip.vmaxf] \tag{1}$$

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`

Aufgabe 20: Perspektivische Projektion mit OpenGL

2. Das Sichtfenster (**Viewport**) kann entsprechend beschränkt werden....
`https://registry.khronos.org/OpenGL-Refpages/gl4/html/glViewport.xhtml`

```
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. 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`

Aufgabe 20: Perspektivische Projektion mit OpenGL

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

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);
glDepthFunc(GL_LEQUAL);
```