Übungen - Bildgenierung

Übung 07.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 10, 2024

Table of Contents

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.

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.

Phong Gleichung:

I_{total} = ambient + Entfernungsabhängige Dämpfung × (diffuse Reflex + Winkelaghängiger);

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. 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
```

- I_a = ambient = lightAmbient * materialAmbient. (7.2.1)

Ambientes Licht: Fertig!

Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

Was ist mit der **diffusen Reflexion**?

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? (7.2.2)

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? (7.2.2)

$I_d =$ <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? (7.2.2)
$I_d =$ <normale, l > * lightDiffuse * materialDiffuse.
Fast: "Im Falle eines negativen Skalarproduktes ist der entsprechende Lichtanteil auf Null zu setzen, um keine negativen Lichtintensitäten zu erzeugen."

Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Diffuse Reflexion.

Wie berechnet man die Intensität von diffus reflektiertem Licht? (7-18)
I_d = <normale, l> * lightDiffuse * materialDiffuse

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

Diffusen Reflexion: Fertig.

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**? (7.2.3)

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**? (7.2.3)
Wie können wir *s* rechnnen? (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**? (7.2.3)
Wie können wir s rechnnen? (7-19)

```
Vec3D s = 2 * skalarprod(normale, l) * normale - l1; $\qquad$
↪  (7-19)
```

$I_w = (v^T \cdot s)^{v_k} * I_l * R$
Wir kennen schon s, und die andere Variablen?

Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

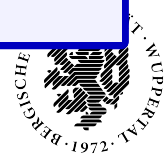
winkelabhängige Reflexion

Was ist mit der **winkelabhängige Reflexion**? 7.2.3
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

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;
}
```

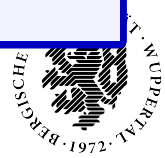
winkelabhängige Reflexion: Fertig!

Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

Entfernungsabhängige Dämpfung

Was ist mit der **Entfernungsabhängige Dämpfung**? (7.2.4)
Einfach, wir haben alle $c-$ Werte.

```
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);

}
```

Entfernungsabhängige Dämpfung: Fertig!

Aufgabe 18: Färbung – Beleuchtungsmodell nach Phong

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

  // Ambient
  VecRGB ambient = lightAmbient * materialAmbient;

  // Diffus
  VecRGB diffuse = lambert * lightDiffuse * materialDiffuse;

  // winkelabhängige Reflexion
  VecRGB specular = cospsiny * lightSpecular * materialSpecular;

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

}
// FERTIG!
```

Table of Contents

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

Schauen wir uns zuerst das Rahmenprogramm 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.

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

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;
  /*...
   cop;     tgt;    vup;
  ...*/
  vector<Vec3D> cops(npics); // "look at"
  vector<Vec3D> vrps(npics); // Kamera Pos
}
```

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

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)
    {...
}
```

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?

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?

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

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?

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 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;
    }
```

Aufgabe 19: Rundflug um den Eiffelturm

Film aus 100 Einzelbildern zuzammen.

Gut! Jetz, 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 zuzammen.

Gut! Jetz, 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

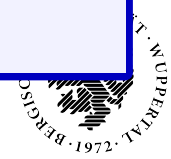
Film aus 100 Einzelbildern zuzammen.

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 zuzammen.

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);

    pics[i] = pic;
  }
  // Fertig!
```

Table of Contents

Table of Contents

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 Initializieren!
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 Initializieren!
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.


BERGISCHE · UNIVERSITÄT · WUPPERTAL · 1972 ·