Aufgabe 9: Lokale Beleuchtungsmodelle von Phong und Blinn

Der Unterschied zwischen beiden Modellen liegt lediglich in der Berechnung des spekularen Anteils, der sich bei Phong aus

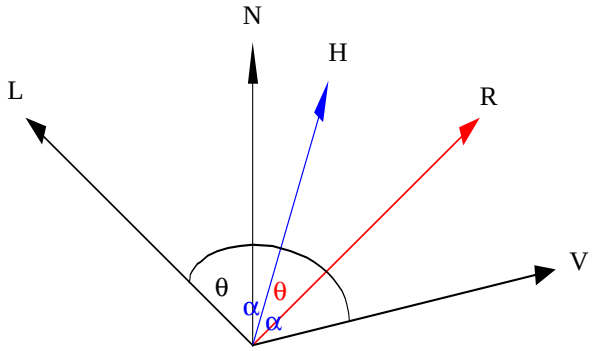
$$I_s \cdot k_s \cdot (R \bullet V)^n$$

errechnet, während Blinn einen sogenannten *halfway vector* verwendet:

$$I_s \cdot k_s \cdot (N \bullet H)^n$$
$$H = \frac{V + L}{2}$$

Anmerkung: Alle Vektoren müssen normiert werden.

a) Zunächst trage ich die Vektoren, die in beiden Beleuchtungsmodellen auftauchen, in eine gemeinsame Zeichnung ein:



Für die Winkel gilt:

$$\alpha = \angle(L,H) = \angle(H,V)$$
$$\theta = \angle(L,N) = \angle(N,R)$$

wobei $\beta = \angle(a,b)$ bedeutet, dass β der von den Vektoren *a* und *b* eingeschlossene Winkel ist.
Nun führe ich einen neuen Winkel δ ein, der von *N* und *H* eingeschlossen wird:

$$\delta = \angle(N,H)$$
$$\delta = \alpha - \theta$$

Da *N* bzw. *H* eine Art „Winkelhalbierende“ darstellen, kann man folgern:

$$\begin{aligned} 2\alpha &= \angle(L,V) \\ &= 2 \cdot (\delta + \theta) \\ &= 2\delta + 2\theta \\ &= 2\delta + \angle(L,R) \\ 2\delta &= \angle(R,V) \end{aligned}$$

Noch einmal kurz wiederholt:

$$\begin{aligned} \delta &= \angle(N,H) \\ 2\delta &= \angle(R,V) \\ 2\cdot\angle(N,H) &= \angle(R,V) \end{aligned}$$

Der Kosinus ist laut Definition zwischen 0° und 90° monoton fallend. Da der von *L* und *V* eingeschlossene Winkel stets kleiner oder gleich 180° ist, sind alle benannten Winkel kleiner gleich 90°. Als Schlussfolgerung:

$$\begin{aligned} 2\cdot\angle(N,H) &= \angle(R,V) \\ \cos\angle(N,H) &\geq \cos\angle(R,V) \\ N\bullet H &\geq R\bullet V \\ (N\bullet H)^n &\geq (R\bullet V)^n \qquad \textit{für } n>1 \end{aligned}$$

Der Unterschied macht sich dadurch bemerkbar, dass die durch spekulare Beleuchtung bewirkte Lichtintensität beim Blinn-Modell größer als beim Phong-Modell ist. Der visuelle Effekt ist ein stärkerer Spotlight-Fleck auf den illuminierten Szenenobjekten.
Im Internet fand ich die Aussage, dass das Blinn-Modell besser mit den experimentell ermittelten Beleuchtungsintensitäten übereinstimmt.

b) Unter Verwendung eines distant lights und einer Parallelprojektion sind die Vektoren *L* und *V* konstant. Da sich der *halfway vector H* direkt aus ihnen berechnet, ist auch er für die gesamte Szene konstant. Für jede Oberfläche muss in diesem Fall nur noch das Skalarprodukt von *N* und *H* gebildet werden.
Im Phong-Beleuchtungsmodell ist stets die gesamte Formel

$$R \bullet V = \left(2 \cdot N \cdot \left(N \bullet L\right) - L\right) \bullet V$$

zu evaluieren.
Betrachtet man den Rechenaufwand im allgemeinen Fall, so stellt man im R^3 fest:

Phong:

Ausdruck	**Operationen**
$N \bullet L$	2 Additionen, 3 Multiplikationen
$2 \cdot \left(N \bullet L\right)$	2 Additionen, 4 Multiplikationen
$2 \cdot N \cdot \left(N \bullet L\right)$	2 Additionen, 7 Multiplikationen
$2 \cdot N \cdot \left(N \bullet L\right) - L$	5 Additionen, 7 Multiplikationen
$\left(2 \cdot N \cdot \left(N \bullet L\right) - L\right) \bullet V$	7 Additionen, 10 Multiplikationen

Blinn:

Ausdruck	**Operationen**
$V + L$	3 Additionen
$\frac{V + L}{2}$	3 Additionen, 3 Multiplikationen
$N \bullet \frac{V + L}{2}$	5 Additionen, 6 Multiplikationen

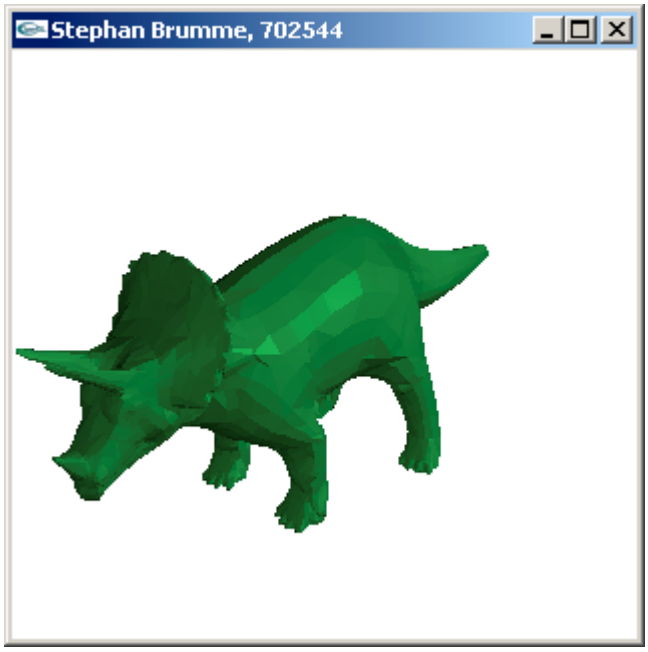
Im Blinn-Modell sind deutlich weniger Rechenoperationen notwendig (insbesondere die aufwändigen Multiplikationen haben sich nahezu halbiert), so dass die Illumination mit weniger Rechenzeit durchführbar ist.

Aufgabe 10: Phong-Beleuchtungsmodell

Um das Programm bedienen zu können, sind diese Tasten notwendig:

Taste	Aktion
ESC	Programm beenden
Leertaste	Objekt drehen
+	in die Szene hineinzoomen
-	aus der Szene herauszoomen
l	Drahtgittermodell
f	Oberflächen zeichnen
1	OpenGL-Beleuchtung (nach Blinn)
2	eigene Phong-Beleuchtung
c	Culling an/aus

Standardmäßig ist die Phong-Beleuchtung aktiv und alle Oberflächen werden gezeichnet, wie der Screenshot zeigt:



Es existieren zwei Lichtquellen, eine bei (-1, 2, 6), die andere bei (9, 4, 1). Das Modell ist im Ursprung zentriert, die Kamera steht im Punkt (0, 3, 7).

Das Phong-Modell beruht auf der Gleichung:

$$I = I_a k_a + \sum_{i=1}^{N} f_{att_i} \cdot \left(I_d k_d (N \bullet L_i) + I_s k_s (R_i \bullet V)^n \right)$$

Ich verzichte bewusst auf farbige Lichtquellen, da dies auch nicht explizit in der Aufgabenstellung gefordert war. Da ich somit nur eine Helligkeit ermittle, kann man sehr schnell die anzuzeigende Farbe berechnen (zur Sicherheit wird noch geclampt):

```
// Material color
Color c = ctx_->material->color;
c *= dIntensity;

// color per vertex
glColor3f(min(c[0],1), min(c[1],1), min(c[2],1));
```

Die eigentliche Umsetzung der Phong-Gleichung geschieht auf folgendem Wege:

```
    // define intensity variables
    double dIntensityAmbient  = 0;
    double dIntensityDiffuse  = 0;
    double dIntensitySpecular = 0;

    // ambient intensity
    dIntensityAmbient = Light::ambient * ctx_->material->ka;

    // N = normal
    const Vector N_norm = normal.normalized();
    // V = view vector
    const Vector V = *(ctx_->camera) - point;
    const Vector V_norm = V.normalized();

    // process all light sources (at most 8)
    for (int nLight=0; nLight < MAXLIGHTSOURCES; nLight++)
    {
        // get a single light source
        Light* light = ctx_->light[nLight];
        if (light == NULL)
            break;

        // L = light vector
        const Vector L = light->pos - point;
        const Vector L_norm = L.normalized();

        // N*L
        const double NdotL = max(dotProduct(N_norm, L_norm), 0);
        // R*V = (2*N*(N*L)-L)*V
        const double RdotV = max(dotProduct((2*NdotL*N_norm - L_norm).normalized(),V_norm),0);

        // attenuation
        const double dDistance = abs(L);
        double dAttenuation = 1 / (light->constant + dDistance*light->linear
                                   + dDistance*dDistance*light->quadric);
        if (dAttenuation > 1)
            dAttenuation = 1;
        // attenuation*I
        const double dAttI  = dAttenuation * light->intensity;

        // diffuse intensity
        dIntensityDiffuse  += dAttI * ctx_->material->kd * NdotL;
        // specular intensity
        dIntensitySpecular += dAttI * ctx_->material->ks * pow(RdotV, ctx_->material->n);
    }

    const double dIntensity = dIntensityAmbient + dIntensityDiffuse + dIntensitySpecular;

    // Material color
    Color c = ctx_->material->color;
    c *= dIntensity;

    // color per vertex
    glColor3f(min(c[0],1), min(c[1],1), min(c[2],1));
```

Um meine Implementierung testen zu können, habe ich das von OpenGL verwendete Blinn-Modell als Vergleichsmaßstab herangezogen. Damit die Lichtquellen die gleichen Parameter haben, definierte ich:

```
const float specular[]  = { 0, ctx_->light[0]->intensity, 0, 0 };
    glMaterialfv(GL_FRONT, GL_SPECULAR,  specular);
    glMaterialf (GL_FRONT, GL_SHININESS, ctx_->material->n);

    const float lightPosition0[] = { ctx_->light[0]->pos[0],
                                     ctx_->light[0]->pos[1],
                                     ctx_->light[0]->pos[2], 0 };
    const float lightDiffuse0[] = { 0, ctx_->light[0]->intensity, 0, 0 };
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE,  lightDiffuse0);
    glEnable(GL_LIGHT0);

    const float lightPosition1[] = { ctx_->light[1]->pos[0],
                                     ctx_->light[1]->pos[1],
                                     ctx_->light[1]->pos[2], 0 };
```

```
    const float lightDiffuse1[] = { 0, ctx_->light[1]->intensity, 0, 0 };
    glLightfv(GL_LIGHT1, GL_POSITION, lightPosition1);
    glLightfv(GL_LIGHT1, GL_DIFFUSE,  lightDiffuse1);
    glEnable(GL_LIGHT1);

    glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);
```

Leider war es noch notwendig, auch die Rotation der Vektoren zu übernehmen:

```
inline Vector rotateYaxis(const Vector& vec, float angle)
{
    Vector result;
    const float sinus   = sin(angle/180*3.1415926);
    const float cosinus = cos(angle/180*3.1415926);
    result[0] = vec[0]*cosinus + vec[2]*sinus;
    result[1] = vec[1];
    result[2] =-vec[0]*sinus   + vec[2]*cosinus;
    return result;
}

void CGIllumination::onDraw() {
    // Loesche den Farb- und Tiefenspeicher
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
    glBegin(GL_TRIANGLES);
    for (int i=0; i<size_; i++)
    {
        const Vector n = rotateYaxis(tris_[i].triangleNormal(), rotate_).normalized();
        glNormal3dv(n.rep());
        for (int k = 0; k < 3; k++) {
            const Vector v = rotateYaxis(tris_[i].getVertex(k), rotate_) * zoom_;
            setPhongIllumination(v,n);
            glVertex3dv(v.rep());
        }
    }
    glEnd();
    glPopMatrix();

    // Nicht vergessen! Front- und Back-Buffer tauschen:
    swapBuffers();
}
```

Aufgabe 11: Materialeinstellung in OpenGL

Die Steuerung des Programms durch den Benutzer ist leider nicht allzu umfangreich:

Taste	Aktion
ESC	Programm beenden
Leertaste	Objekt drehen
+	in die Szene hineinzoomen
-	aus der Szene herauszoomen

Generell sind die linken Tori dunkler als die rechten, wobei dies von der groben Richtung der Lichtquelle abhängt, deshalb wird horizontal der diffuse Anteil erhöht. Von oben nach unten nimmt die Größe eines reflektierenden Punktes und die Grundhelligkeit zu. Man kann diesen Effekt bei den Materialeinstellungen durch Verringern des spekularen Exponenten erreichen.
Eine gewisse Grundhelligkeit wird durch ein konstanten ambienten Lichtanteil von 0,5 erreicht.

Die einzige Lichtquelle befindet sich bei (-3, 3, 0):

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
static const float lightPosition0[] = { -3,
                                         3,
                                         0, 0 };
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition0);
```

Weil es nicht notwendig ist, dass diese Lichtquelle in irgendeinem Parameter verändert wird, habe ich sie gleich in den Programminitialisierungscode verschoben und die Funktion `setLight` ohne Funktionalität gelassen.

Die Materialeigenschaften werden in `setMaterial` bestimmt. Die ambiente Reflexion beschränkt sich auf einen konstanten Rotanteil von 50%. Das diffuse Licht ist ebenfalls nur im Rotbereich sichtbar, hier versuche ich, horizontal (d.h. mit Hilfe von `x`) den Bereich 0 bis 1 zu durchlaufen. Das spekulare Licht ist grau (50%), die von mir empirisch ermittelte Formal für den spekularen Exponenten lautet $16 \cdot 2^y$.

```
void CGMaterial::setMaterial(int x, int y) {
    // Hier werden die Materialeigenschaften festgelegt

    // ambient
    float ambient[4];
    ambient[0] = 0.5;
    ambient[1] = ambient[2] = ambient[3] = 0;
    glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);

    // diffuse
    float diffuse[4];
    diffuse[0] = x/(XWindows-1.0);
    diffuse[1] = diffuse[2] = diffuse[3] = 0;
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);

    // specular
    static const float specular[]  = { 0.5, 0.5, 0.5, 0.5 };
    glMaterialfv(GL_FRONT, GL_SPECULAR,  specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 16*pow(2,y));
}
```

Quelltext

Zunächst cgillumination.h und cgillumination.cpp aus Aufgabe 10:

cgillumination.h:

```
//
// Computergraphik II
// Prof. Dr. Juergen Doellner
// Wintersemester 2001/02
//
// Rahmenprogramm zu Aufgabenzettel 1
//

#ifndef CG_ILUM_H
#define CG_ILUM_H

#include "cgapplication.h"
#include "triangle.h"

class Context;

class CGIllumination : public CGApplication {
public:
    CGIllumination();
    virtual ~CGIllumination();

    // Ueberschreibe alle diese Ereignisse:
    virtual void onInit();
    virtual void onDraw();
    virtual void onIdle();
    virtual void onKey(unsigned char key);
    virtual void onSize(unsigned int newWidth, unsigned int newHeight);

private:

    void setPhongIllumination(const Vector& point, const Vector& normal);

    int size_;
    Triangle* tris_;
    Context* ctx_;

    float zoom_;
    float rotate_;

    // state vars
    bool stop_;
    bool phong_;

};

#endif // CG_ILUM_H
```

cgillumination.cpp:

```
//
// Computergraphik II
// Prof. Dr. Juergen Doellner
// Wintersemester 2001/02
//
// Rahmenprogramm zu Aufgabenzettel 1
//

#include "cgillumination.h"
#include <fstream.h>

typedef Vector Camera;
typedef Vector Color;

const MAXLIGHTSOURCES = 8;

class Light {
public:
    Light(const Vector& p, double i) : pos(p),intensity(i) {
        constant = 0.3;
        linear = 0.0;
        quadric = 0.0;
    }

    Vector pos;                 /* position, world coordinate system   */
    double intensity;           /* light source intensity [0,1]        */
    static double ambient;      /* ambient light contribution (global) */

    double constant;            /* constant attenuation                */
    double linear;              /* linear attenuation                  */
    double quadric;             /* quadric attenuation                 */
};

double Light::ambient = 0.2;

class Material {
public:
    Material(double ka_,double kd_,double ks_,double n_, Color c)
        : ka(ka_), kd(kd_), ks(ks_), n(n_), color(c) {}

    double ka;                  /* ambient reflection coefficient    */
    double kd;                  /* diffuse reflection coefficient    */
    double ks;                  /* specular reflection coefficient   */
    double n;                   /* specular shininess                */
    Color color;                /* specular color                    */
};

class Context {
public:
    Light* light[MAXLIGHTSOURCES];            /* max. 8 light sources */
    Camera* camera;
    Material* material;
};


//
//  CGIllumination Application
//

CGIllumination::CGIllumination() {

    stop_=true;
    phong_=true;
    zoom_=0.3;
    rotate_=-30;

    // read triangle data
    ifstream s("triceratops.txt");
    //ifstream s("triangle_small.txt");
    s >> size_;

    tris_ = new Triangle[size_];

    int i;
```

```
    for (i = 0; i < size_; i++)
        s >> tris_[i];

    // init context
    ctx_= new Context();
    ctx_->material = new Material(0.6, 0.4, 0.2, 32.0 , Color(0.4, 0.9, 0.4));
    ctx_->camera = new Camera(0, 3, 7);
    ctx_->light[0] = new Light(Vector( -1, 2, 6), 0.7 );
    ctx_->light[1] = new Light(Vector( 9, 4, 1), 1.0 );
    ctx_->light[2] = 0;
}


CGIllumination::~CGIllumination() {
}

void CGIllumination::onInit() {

    // zu Beginn keine OpenGL Lichtquelle
    glDisable(GL_LIGHTING);

    const float specular[]  = { 0, ctx_->light[0]->intensity, 0, 0 };
    glMaterialfv(GL_FRONT, GL_SPECULAR,  specular);
    glMaterialf (GL_FRONT, GL_SHININESS, ctx_->material->n);

    const float lightPosition0[] = { ctx_->light[0]->pos[0],
                                     ctx_->light[0]->pos[1],
                                     ctx_->light[0]->pos[2], 0 };
    const float lightDiffuse0[] = { 0, ctx_->light[0]->intensity, 0, 0 };
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE,  lightDiffuse0);
    glEnable(GL_LIGHT0);

    const float lightPosition1[] = { ctx_->light[1]->pos[0],
                                     ctx_->light[1]->pos[1],
                                     ctx_->light[1]->pos[2], 0 };
    const float lightDiffuse1[] = { 0, ctx_->light[1]->intensity, 0, 0 };
    glLightfv(GL_LIGHT1, GL_POSITION, lightPosition1);
    glLightfv(GL_LIGHT1, GL_DIFFUSE,  lightDiffuse1);
    glEnable(GL_LIGHT1);

    glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);


    // automatische Normalisierung
//    glEnable(GL_NORMALIZE);

    // Tiefen Test aktivieren
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);
    glClearColor(1.0,1.0,1.0,1.0);

    // Smooth Schattierung aktivieren
    glShadeModel(GL_SMOOTH);

    // Culling
    glCullFace(GL_BACK);

    // Projection
    glMatrixMode(GL_PROJECTION);
    gluPerspective(40.0,1.0,1.0,10.0);

    // LookAt
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(0.0, 3.0, 7.0,  // from (0,3,7)
        0.0, 0.0, 0.0,  // to (0,0,0)
        0.0, 1.0, 0.);  // up

}
void CGIllumination::onSize(unsigned int newWidth,unsigned int newHeight) {
    if((newWidth > 0) && (newHeight > 0)) {
        // Passe den OpenGL-Viewport an die neue Fenstergroesse an:
        glViewport(0, 0, newWidth - 1, newHeight - 1);

        // Passe die OpenGL-Projektionsmatrix an die neue
```

```
        // Fenstergroesse an:
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(40.0,float(newWidth)/float(newHeight),1.0, 10.0);

        // Schalte zurueck auf die Modelview-Matrix
        glMatrixMode(GL_MODELVIEW);
    }
}

void CGIllumination::onKey(unsigned char key) {
    static GLfloat z = 3.;
    switch (key) {
    case 27: { exit(0); break; }
    case '+': { zoom_*= 1.1; break; }
    case '-': { zoom_*= 0.9; break; }
    case 'l' : { glPolygonMode(GL_FRONT_AND_BACK,GL_LINE); break; }
    case 'f' : { glPolygonMode(GL_FRONT_AND_BACK,GL_FILL); break; }
    case '1' : { glEnable(GL_LIGHTING); phong_ = false; break; }
    case '2' : { glDisable(GL_LIGHTING); phong_ = true; break; }
    case 'c' : { if (glIsEnabled(GL_CULL_FACE)) { glDisable(GL_CULL_FACE); } else {
glEnable(GL_CULL_FACE); } break; }
    case ' ' : { stop_ = !stop_; break; }
    }
    onDraw();
}

void CGIllumination::onIdle() {
    if (!stop_) {
        rotate_ -= 2;
        onDraw();
    }
}

inline double max(double x, double y) {
    return (x<y) ? y : x;
}
inline double min(double x, double y) {
    return (x<y) ? x : y;
}

void CGIllumination::setPhongIllumination(const Vector& point, const Vector& normal) {
    // OpenGL's lighting ?
    if (!phong_)
    {
        const Color c = ctx_->material->color;
        glColor3f(c[0], c[1], c[2]);
        return;
    }

    // Phong Beleuchtungsmodell implementieren

    // NOW COMES DA REAL STUFF !!!

    // define intensity variables
    double dIntensityAmbient  = 0;
    double dIntensityDiffuse  = 0;
    double dIntensitySpecular = 0;

    // ambient intensity
    dIntensityAmbient = Light::ambient * ctx_->material->ka;

    // N = normal
    const Vector N_norm = normal.normalized();
    // V = view vector
    const Vector V = *(ctx_->camera) - point;
    const Vector V_norm = V.normalized();

    // process all light sources (at most 8)
    for (int nLight=0; nLight < MAXLIGHTSOURCES; nLight++)
    {
        // get a single light source
        Light* light = ctx_->light[nLight];
        if (light == NULL)
            break;
```

```
        // L = light vector
        const Vector L = light->pos - point;
        const Vector L_norm = L.normalized();

        // N*L
        const double NdotL = max(dotProduct(N_norm, L_norm), 0);
        // R*V = (2*N*(N*L)-L)*V
        const double RdotV = max(dotProduct((2*NdotL*N_norm - L_norm).normalized(), V_norm),
0);

        // attenuation
        const double dDistance = abs(L);
        double dAttenuation = 1 / (light->constant + dDistance*light->linear
                                   + dDistance*dDistance*light->quadric);
        if (dAttenuation > 1)
            dAttenuation = 1;
        // attenuation*I
        const double dAttI  = dAttenuation * light->intensity;

        // diffuse intensity
        dIntensityDiffuse  += dAttI * ctx_->material->kd * NdotL;
        // specular intensity
        dIntensitySpecular += dAttI * ctx_->material->ks * pow(RdotV, ctx_->material->n);
    }

    const double dIntensity = dIntensityAmbient + dIntensityDiffuse + dIntensitySpecular;

    // Material color
    Color c = ctx_->material->color;
    c *= dIntensity;

    // color per vertex
    glColor3f(min(c[0],1), min(c[1],1), min(c[2],1));
}


inline Vector rotateYaxis(const Vector& vec, float angle)
{
    Vector result;
    const float sinus   = sin(angle/180*3.1415926);
    const float cosinus = cos(angle/180*3.1415926);
    result[0] = vec[0]*cosinus + vec[2]*sinus;
    result[1] = vec[1];
    result[2] =-vec[0]*sinus   + vec[2]*cosinus;
    return result;
}

void CGIllumination::onDraw() {
    // Loesche den Farb- und Tiefenspeicher
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


    glPushMatrix();
//  glRotatef(rotate_,0,1,0);
//  glScaled(zoom_, zoom_, zoom_);

    glBegin(GL_TRIANGLES);
    for (int i=0; i<size_; i++)
    {
        const Vector n = rotateYaxis(tris_[i].triangleNormal(), rotate_).normalized();
        glNormal3dv(n.rep());
        for (int k = 0; k < 3; k++) {
            const Vector v = rotateYaxis(tris_[i].getVertex(k), rotate_) * zoom_;
            setPhongIllumination(v,n);
            glVertex3dv(v.rep());
        }
    }
    glEnd();

    glPopMatrix();

    // Nicht vergessen! Front- und Back-Buffer tauschen:
    swapBuffers();
}

// Hauptprogramm
```

```
int main(int argc, char* argv[]) {
    // Erzeuge eine Instanz der Beispiel-Anwendung:
    CGIllumination sample;

    cout << "Tastenbelegung:" << endl
         << "ESC        Programm beenden" << endl
         << "Leertaste  Objekt drehen" << endl
         << "+          Hineinzoomen" << endl
         << "-          Herauszoomen" << endl
         << "l          Drahtgittermodell" << endl
         << "f          Oberflaechen zeichnen" << endl
         << "1          OpenGL-Beleuchtung (Blinn-Modell)" << endl
         << "2          eigene Phong-Beleuchtung" << endl
         << "c          Rueckseiten zeichnen an/aus" << endl;

    // Starte die Beispiel-Anwendung:
    sample.start("Stephan Brumme, 702544");
    return(0);
}
```

Für Aufgabe 11 habe ich nur cgmaterial.cpp verändert:

cgmaterial.cpp:

```
//
// Computergraphik II
// Prof. Dr. Juergen Doellner
// Wintersemester 2001/02
//
// Rahmenprogramm zu Aufgabenzettel 4
//

#include "material.h"

CGMaterial::CGMaterial() {
    run_ = false;
    zoom_ = 0.8;
}

CGMaterial::~CGMaterial( ) {
}

void CGMaterial::setLight(int x, int y) {
    // Hier werden die Lichtquellen definiert
}

void CGMaterial::setMaterial(int x, int y) {
    // Hier werden die Materialeigenschaften festgelegt

    // ambient
    float ambient[4];
    ambient[0] = 0.5;
    ambient[1] = ambient[2] = ambient[3] = 0;
    glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);

    // diffuse
    float diffuse[4];
    diffuse[0] = x/(XWindows-1.0);
    diffuse[1] = diffuse[2] = diffuse[3] = 0;
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);

    // specular
    static const float specular[]  = { 0.5, 0.5, 0.5, 0.5 };
    glMaterialfv(GL_FRONT, GL_SPECULAR,  specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 16*pow(2,y));
}

void CGMaterial::setSmallViewport(int x, int y) {
    int sizeX = (width_/XWindows);
    int sizeY = (height_/YWindows);
    glViewport(x*sizeX, y*sizeY, sizeX - 1, sizeY - 1);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(36.0,float(sizeX)/float(sizeY),2.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

void CGMaterial::drawScene() {

    glPushMatrix();
    glScaled(zoom_, zoom_, zoom_);
    glRotatef(70.0, 1., 0., 0.);

    glColor4f(1,0,0,0);

    static GLuint cache = 0;
    if (cache == 0) {
        cache = glGenLists(1);
        glNewList(cache, GL_COMPILE_AND_EXECUTE);

        // Hinweis: Falls das Programm zu langsam läuft, kann man die Tessellation
        // des Torus durch Verringern des dritten und vierten Parameters vergröbern.
        // Dadurch leidet aber die Darstellungsqualität.
        glutSolidTorus(0.5, 1.0, 40, 40);
```

```
        //glutSolidTeapot(1.0);

        glEndList();
    } else {
        glCallList(cache);
    }

    glPopMatrix();
}

void CGMaterial::onInit() {

    // automatische Normalisierung
    glEnable(GL_NORMALIZE);

    // Tiefen Test aktivieren
    glEnable(GL_DEPTH_TEST);

    // Beleuchtung aktivieren
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    static const float lightPosition0[] = { -3,
                                            3,
                                            0, 0 };
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition0);
//    glEnable(GL_COLOR_MATERIAL);
//    glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);

    // Smooth Schattierung aktivieren
    glShadeModel(GL_SMOOTH);

    // LookAt
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(0.0, 0.0, 4.0,  // from (0,0,4)
        0.0, 0.0, 0.0,  // to (0,0,0)
        0.0, 1.0, 0.);  // up

    glClearColor(1.0, 1.0, 1.0, 1.0);
}
void CGMaterial::onSize(unsigned int newWidth,unsigned int newHeight) {
    width_ = newWidth;
    height_ = newHeight;
    glViewport(0, 0, width_ - 1, height_ - 1);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0,float(width_)/float(height_),2.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

void CGMaterial::onKey(unsigned char key) {
    switch (key) {
    case 27: { exit(0); break; }
    case '+': { zoom_*= 1.1; break; }
    case '-': { zoom_*= 0.9; break; }
    case ' ': { run_ = !run_; break; }
    }
    onDraw();
}

void CGMaterial::onIdle() {
    if (run_) {
        glRotatef(5,0.1,1,0.2);
        onDraw();
    }
}

void CGMaterial::onDraw() {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    for (int x=0; x<XWindows; x++)
        for (int y=0; y<YWindows; y++) {
//            setLight(x, y);
            setMaterial(x, y);
```

```
            setSmallViewport(x, y);
            drawScene();
            glFlush();
    }

    // Front- und Back-Buffer tauschen:
    swapBuffers();
}

// Hauptprogramm
int main(int argc, char* argv[]) {
    // Erzeuge eine Instanz der Beispiel-Anwendung:
    CGMaterial sample;

    cout << "Tastenbelegung:" << endl
         << "ESC        Programm beenden" << endl
         << "Leertaste  Objekt drehen" << endl
         << "+          Hineinzoomen" << endl
         << "-          Herauszoomen" << endl;

    // Starte die Beispiel-Anwendung:
    sample.start("Stephan Brumme, 702544", true, 550, 550);
    return(0);
}
```