Bericht zur 03 Beleuchtung I

Marcus Bätz, Andreas Kiauka, Robert Dziuba

29. November 2015

1 Aufgabenstellung

Es sollen Beleuchtungen und Materialien in den Raytracer implementiert werden. Für die Umsetzung nutzen wir die UML Diagramme aus der Übungsaufgabe.

1.1 Light

Es soll eine abstrakte Oberklasse *Light* erstellt werden, von der wir *PointLight*, *DirectionalLight* und *SpotLight* ableiten.

1.2 Material

Des weiteren soll eine abstrakte Oberklasse *Material* erstellt werden, von der wir *SingleColorMaterial*, *LambertMaterial* und *PhongMaterial* ableiten.

1.3 Änderungen

Damit unser Licht und Material in den Raytracer integriert werden kann, müssen Änderungen an der *World-Klasse* und den *Geometry-Klassen* vorgenommen werden

1.4 Demo-Szene

Schlussendlich implementieren wir die beschriebene Demo-Szenen.

2 Lösungstrategien

Die Klassen wurden gerecht aufgeteilt und von jedem einzelnen in seiner GIT-Branch, um gegenseitige Konflikte zu vermeiden, bearbeitet. Damit wir keine Probleme mit fehlenden Klassen hatten, haben wir vorher auf dem Master-Branch alle oben genannten Klassen als Dummy-Klassen erzeugt. Die fertigen Klassen wurden danach wieder auf den Master gemerged.

3 Implementierung

3.1 Light

Diese abstrakte Klasse bekommt die Farbe des Lichts übergeben. Außerdem implementiert sie die beiden abstrakten Methoden *illuminates* und *directionFrom*. *illuminates* gibt uns zurück, ob ein getroffener Punkt leuchtet oder nicht. *directionFrom* erzeugt einen Vektor, der vom getroffenen Punkt aus auf die Lichtquelle zeigt.

```
1  public abstract class Light extends Element implements
       Serializable {
2
3      public final Color color;
4
5      public Light(Color color) {
6          this.color = color;
7      }
8
9      public abstract boolean illuminates(Point3 point);
10
11     public abstract Vector3 directionFrom(Point3 point);
12 }
```

3.2 DirectionalLight

Die Klasse *DirectionalLight* stellt die Sonne dar. Sie wird definiert als unendlich weit entfernte Lichtquelle, bei der die Richtung zur Lichtquelle überall gleich ist. Daher benötigen wir für die Berechnung des Lichts nur die Richtung der Lichtstrahlen, den wir schon im Konstruktor normalisieren und dann umdrehen.

```
1   public Vector3 directionFrom(Point3 point) {
2         return direction.mul(-1);
3     }
```

3.3 PointLight

Beim *PointLight* wird eine Lampe imitiert. Hierbei gehen alle Strahlen gleichmäßig in alle Richtung von einem Punkt aus. Um die Richtung der Lichtquelle zu berechnen, müssen wir den Punkt von der Position subtrahieren und normalisieren.

```
1   public Vector3 directionFrom(Point3 point) {
2         return position.sub(point).normalized();
3     }
```

3.4 SpotLight

Das Spotlight ist eine Spezialisierung des Pointlights. Es simuliert einen gerichteten Strahler, der eine feste Position hat und nur aus einer Richtung, innerhalb eines bestimmten Winkels, strahlt. Die Methode *directionFrom* ist gleich der vom *PointLight*. Nur in der Methode *illuminates* muss zusätzlich berechnet werden, ob ein getroffener Punkt wirklich im Lichtkegel liegt.

```
1   public boolean illuminates(final Point3 point) {
2         return Math.acos(direction.dot(directionFrom(point)
              .mul(-1))) <= halfAngle;
3     }
```

3.5 Material

Damit unsere geometrischen Figuren auch verschiedene "Materialien" wie z.B. Matt oder Glänzend, simulieren können, müssen wir diese verschiedenen Materialien implementieren. Die abstrakte Basisklasse *Material* hat eine Methode *colorFor*, dessen Implementierungen die Farbe für ein Hit-Objekt zurückgibt. Die Klasse wird mit einer Farbe initialisiert.

```
1  public abstract class Material implements Serializable{
2
3      public final Color diffuse;
4
5      public Material(final Color diffuse) {
6          this.diffuse = diffuse;
7      }
8
9      public abstract Color colorFor(Hit hit, World world);
10 }
```

3.6 SingleColorMaterial

Das *SingleColorMaterial* entspricht dem bisherigen Verhalten unserer Geometrien. Die in dem Konstruktor übergebene Farbe wird einfach von colorFor zurück gegeben, unabhängig von Lichtquellen oder der Normalen.

```
1  public Color colorFor(Hit hit, World world) {
2          return diffuse;
3      }
```

3.7 LambertMaterial

Die *LambertMaterial*-Klasse stellt eine Oberfläche wie Kreide dar. Für den perfekt diffus reflektierenden Körper rechnen wir zu unserer Körperfarbe die Farbe des Lichts hinzu. Dabei muss geprüft werden, ob der getroffene Punkt einen Vektor besitzt der eine unserer Lichtquellen reflektiert. Natürlich überprüfen wir vorher, ob unser getroffener Punkt auch wirklich im Lichtkegel liegt. Zur Aufhellung der gesamten Szene wird zusätzliche unserem Material ein "ambientes" Licht hinzugefügt.

```
1  public Color colorFor(Hit hit, World world) {
```

```
2           Color c = new Color(0,0,0);
3           for(Light light: world.lights){
4               final Point3 p = hit.ray.at(hit.t);
5               if(light.illuminates(p)){
6                   c= c.add(light.color.mul(diffuse).mul(Math.
                        max(0, hit.n.dot(light.directionFrom(p))
                        )));
7               }
8           }
9
10          return diffuse.mul(world.ambientLight).add(c);
11      }
```

3.8 PhongMaterial

Das Phong Material stellt ein Material für eine glatte, porzellanähnliche Oberfläche mit Glanzpunkt dar. Man kann sich den Glanzpunkt wie bei Billard Kugeln vorstellen. Bei der spiegelnden Reflexion wird das Licht in einer bestimmten Umgebung von der Oberfläche reflektiert. Die Lichtstärke des reflektierten Lichtes ist vom Einfallswinkel des Lichtstrahls der Punktlichtquelle, von dem Phong-Exponenten sowie der Oberfläche und vom Blickwinkel des Beobachters der Szene abhängig.

```
1   public Color colorFor(final Hit hit, final World world) {
2           Color basicColor = new Color(0, 0, 0);
3           final Vector3 e = hit.ray.d.mul(-1).normalized();
4           final Point3 h = hit.ray.at(hit.t);
5
6           for (Light light : world.lights) {
7               Vector3 l = light.directionFrom(h);
8               Vector3 rl = l.reflectedOn(hit.n);
9               if (light.illuminates(h)) {
10                  basicColor = basicColor.add(
11                          light.color.mul(diffuse)
12                                  .mul(Math.max(0, hit.n.dot(
                                        l.normalized())))
13                                  )
14                  ).add(
15                          specular
16                                  .mul(light.color)
17                                  .mul(Math.pow(
```

```
18                                                  Math.max(0,rl.dot(e
                                                        )),exponent)
19                                          )
20                      );
21                  }
22              }
23              return diffuse.mul(world.ambientLight).add(
                    basicColor);
24      }
```

3.9 Änderungen am Hit-Objekt, der Geometry-Klasse und der World-Klasse

Im Hit-Objekt wurde die Normale des Schnittpunkts hinzugefügt. Des weiteren wird nun in unseren Geometry-Klasse statt der Farbe die Material-Klasse übergeben.

Der Konstruktor für das *Triagle* wurde zusätzlich um die Normalen aller drei Eckpunkte erweitert. Werden diese nicht mit angegeben berechnen wir drei Standard-Normalen. Die Hit-Methode verrechnet nun außerdem die Normale jeder Ecken mit seiner baryzentrischen Koordinate und addiert sie dazu. Da alle 3 Koordinaten zusammen 1 ergeben, muss die neue Normale nicht mehr normalisiert werden.

```
1  public Hit hit(final Ray r) {
2      final Vector3 v = new Vector3(a.x - r.o.x, a.y - r.
           o.y, a.z - r.o.z);
3      final Mat3x3 m = new Mat3x3(
4              a.x - b.x, a.x - c.x, r.d.x,
5              a.y - b.y, a.y - c.y, r.d.y,
6              a.z - b.z, a.z - c.z, r.d.z
7      );
8      final double detA = m.determinant;
9      final double detA1 = m.col1(v).determinant;
10     final double beta = detA1 / detA;
11
12     if (beta >= 0 && beta <= 1) {
13
14         final double detA2 = m.col2(v).determinant;
15         final double gamma = detA2 / detA;
16
17         if ( gamma > 0 && beta + gamma <= 1) {
18             final double detA3 = m.col3(v).determinant;
19             final double t = detA3 / detA;
20             if (t > 0){
21                 Normal3 n = na.mul(1- beta - gamma).add
                       (nb.mul(beta)).add(nc.mul(gamma));
22                 return new Hit(t,n, r, this);
23             }
24         }
25     }
26     return null;
27 }
```

Zusätzlich werden in unserem World-Objekt alle hinzugefügten Lichtquellen in einer ArrayList gespeichert, die, wie oben schon in den unterschiedlichen Materialien gezeigt, bei jedem Materialkontakt durchlaufen wird.

4 Probleme bei der Bearbeitung

Bei der Bearbeitung der Aufgabenstellung direkt traten kaum Probleme auf. Aber durch die zunehmende Komplexität unseres Programms ist eine Fehlerbehandlung immer schwieriger. Daher sollte unbedingt das einwandfreie Funktionieren der Basisklassen garantiert sein, damit man sich bei den weiteren Berechnungen darauf verlassen kann.