Protokol Übung 05 - Szenengraph

Johann Roehl, Bodo Wissemann, Philip Zuschlag

1 Zum Inhalt dieses Protokols

Das Protokoll behandelt die Implementierung die Transformation der verschiedenen Objekte. Hierfür wurden folgende Klassen neu implementiert:

- Node
- Transform
- Matrix4x4

Weiterhin wurden einige Klassen überarbeitet und um benötigte Felder ergänzt.

2 Node

Die Klasse Node ist eine Art Geometrie. Sie bekommt ein Transform-Objekt mit den Methoden Scale, translate(Point3), rotateX,-Y,-Z und eine ArrayList<Geometry> übergeben. Alle in der Liste enthaltenen Geometrien werden über die am Transform-Objekt aufgerufenen Methoden verändert. Es werden alle Hit-Methoden dieser Geometrien aufgerufen und das kleinste t von allen Geometrien gesucht. Hieraus wird der neue Hit gebildet und zurückgegeben.

```java
package geometries;

import ...

public class Node extends Geometry {

    private final Transform trans;
    public ArrayList<Geometry> geos;

    public Node(Transform trans, ArrayList<Geometry> geos) throws IllegalArgumentException {
        this.trans = trans;
        this.geos = geos;

    }

    @Override
    public Hit hit(Ray ray) throws IllegalArgumentException {
        final Ray transformedRay = trans.mul(ray); // gibt transformierten Ray
                                                   // zurück
        double t = Double.MAX_VALUE;
        Hit hitLow = null;

        for (final Geometry g : geos) {

            final Hit hit = g.hit(transformedRay);
            if (hit != null) {
                if (hit.t < t && hit.t > 0.0001) {
                    t = hit.t;
                    hitLow = hit;
                }
            }

        }
        if (hitLow != null) {

            return new Hit(hitLow.t, ray, hitLow.geo, trans.mul(hitLow.n));
        }
        return null;
    }

}
```

Abbildung 1: *Die Klasse Node*

3 Transform

Die Klasse Transform beinhaltet alle Methoden (translate, scale, rotateX, rotateY, rotateZ) mit denen die Geometrien verändert werden können.

```java
public Transform translate(Point3 p) {
    final Transform t = new Transform(new Mat4x4(

    1, 0, 0, p.x, 0, 1, 0, p.y, 0, 0, 1, p.z, 0, 0, 0, 1

    ), new Mat4x4(

    1, 0, 0, -p.x, 0, 1, 0, -p.y, 0, 0, 1, -p.z, 0, 0, 0, 1

    ));

    return new Transform(n.mul(t.n), t.i.mul(i));
}

// ____________________________________________________________
// inverse Skalierungssmatrix:
// 1/x 0 0 0
// 0 1/y 0 0
// 0 0 1/z 0
// 0 0 0 1
public Transform scale(double x, double y, double z) {
    final Transform t = new Transform(new Mat4x4(x, 0, 0, 0, 0, y, 0, 0, 0, 0, z, 0, 0, 0, 0, 1),

    new Mat4x4(1.0 / x, 0, 0, 0, 0, 1.0 / y, 0, 0, 0, 0, 1.0 / z, 0, 0, 0, 0, 1));

    return new Transform(n.mul(t.n), t.i.mul(i));
}

// ____________________________________________________________
// inverse Rotatioinsmatrix_X:
// 1 0 0 0
// 0 cos(a) sin(a) 0
// 0 -sin(a) cos(a) 0
// 0 0 0 1
public Transform rotateX(double angle) {
    final Transform t = new Transform(new Mat4x4(

    1, 0, 0, 0, 0, Math.cos(angle), -Math.sin(angle), 0, 0, Math.sin(angle), Math.cos(angle), 0, 0, 0, 0, 1

    ), new Mat4x4(

    1, 0, 0, 0, 0, Math.cos(angle), Math.sin(angle), 0, 0, -Math.sin(angle), Math.cos(angle), 0, 0, 0, 0, 1

    ));

    return new Transform(n.mul(t.n), t.i.mul(i));
}

// ____________________________________________________________
// inverse Rotatioinsmatrix_Y:
// cos(a) 0 -sin(a) 0
// 0 1 0 0
// sin(a) 0 cos(a) 0
// 0 0 0 1
public Transform rotateY(double angle) {
    final Transform t = new Transform(new Mat4x4(Math.cos(angle), 0, Math.sin(angle), 0, 0, 1, 0, 0, -Math.sin(angle), 0,
            Math.cos(angle), 0, 0, 0, 0, 1), new Mat4x4(Math.cos(angle), 0, -Math.sin(angle), 0, 0, 1, 0, 0, Math.sin(angle), 0,
            Math.cos(angle), 0, 0, 0, 0, 1));
    return new Transform(n.mul(t.n), t.i.mul(i));
}

// ____________________________________________________________
// inverse Rotatioinsmatrix_Z:
// cos(a) sin(a) 0 0
// -sin(a) cos(a) 0 0
// 0 0 1 0
// 0 0 0 1
public Transform rotateZ(double angle) {
    final Transform t = new Transform(new Mat4x4(Math.cos(angle), -Math.sin(angle), 0, 0, Math.sin(angle), Math.cos(angle), 0, 0, 0, 0,
            1, 0, 0, 0, 0, 1), new Mat4x4(Math.cos(angle), Math.sin(angle), 0, 0, -Math.sin(angle), Math.cos(angle), 0, 0, 0, 0, 1, 0,
            0, 0, 0, 1));

    return new Transform(n.mul(t.n), t.i.mul(i));
}
```

Abbildung 2: *Methoden der Klasse Transform*

4 Matrix4x4

Die Matrix4x4 ist die Transformationsmatrix - sie enthält Methoden zum verändern der Vector3, Point3 und Mat4x4 Objekte

```
public Vector3 mul(Vector3 v) {
    return new Vector3(

    v.x * m11 + v.y * m12 + v.z * m13,

    v.x * m21 + v.y * m22 + v.z * m23,

    v.x * m31 + v.y * m32 + v.z * m33);
}

// ____________________________________________________

public Point3 mul(Point3 p) {
    return new Point3(

    p.x * m11 + p.y * m12 + p.z * m13 + m14,

    p.x * m21 + p.y * m22 + p.z * m23 + m24,

    p.x * m31 + p.y * m32 + p.z * m33 + m34

    );
}

// ____________________________________________________

public Mat4x4 mul(Mat4x4 m) {
    return new Mat4x4(m11 * m.m11 + m12 * m.m21 + m13 * m.m31 + m14 * m.m41, m11 * m.m12 + m12 * m.m22 + m13 * m.m32 + m14 * m.m42, m11
            * m.m13 + m12 * m.m23 + m13 * m.m33 + m14 * m.m43, m11 * m.m14 + m12 * m.m24 + m13 * m.m34 + m14 * m.m44, m21 * m.m11 + m22
            * m.m21 + m23 * m.m31 + m24 * m.m41, m21 * m.m12 + m22 * m.m22 + m23 * m.m32 + m24 * m.m42, m21 * m.m13 + m22 * m.m23 + m23
            * m.m33 + m24 * m.m43, m21 * m.m14 + m22 * m.m24 + m23 * m.m34 + m24 * m.m44, m31 * m.m11 + m32 * m.m21 + m33 * m.m31 + m34
            * m.m41, m31 * m.m12 + m32 * m.m22 + m33 * m.m32 + m34 * m.m42, m31 * m.m13 + m32 * m.m23 + m33 * m.m33 + m34 * m.m43, m31
            * m.m14 + m32 * m.m24 + m33 * m.m34 + m34 * m.m44, m41 * m.m11 + m42 * m.m21 + m43 * m.m31 + m44 * m.m41, m41 * m.m12 + m42
            * m.m22 + m43 * m.m32 + m44 * m.m42, m41 * m.m13 + m42 * m.m23 + m43 * m.m33 + m44 * m.m43, m41 * m.m14 + m42 * m.m24 + m43
            * m.m34 + m44 * m.m44);
}

// ____________________________________________________

public Mat4x4 transposed() { // Zeilen und Spalten vertauscht
    return new Mat4x4(

    m11, m21, m31, m41, m12, m22, m32, m42, m13, m23, m33, m43, m14, m24, m34, m44

    );
}
```

Abbildung 3: *Methoden der Klasse Mat4x4*

5 Weitere Anpassungen

Alle Konstruktoren der Materialien wurden derart verändert, dass jeweils nur Standard-Objekte am Nullpunkt des Koordinatensystems erzeugt werden. Alle weiteren Anpassungen der Objekte werden nun über Transformationen realisiert.

5.1 Texturierungen

Um Texturierungen zu ermöglichen wurden die Konstruktoren der Materialien so umgeschrieben, dass Texturen an Stelle der Color Objekte übergeben werden.
Weitere Anpassungen:

- SingleColor-Texture
  - Enthält simple Farben. Für jedes Pixel der Geometrie wird wie gehabt die Farbe ausgerechnet, alle Objekte sehen aus wie vorher.
- Imagetexture
  - Pfad zu Bild wird übergeben, die Pixel des Bildes zu relativen Koordinaten umgerechnet und auf die Geometrien gelegt
- InterpolatedTexture
  - Methode zur Verbesserung der Bildqualität. Es werden Mittelwerte des mittleren Pixels und der umliegenden Pixel gebildet.

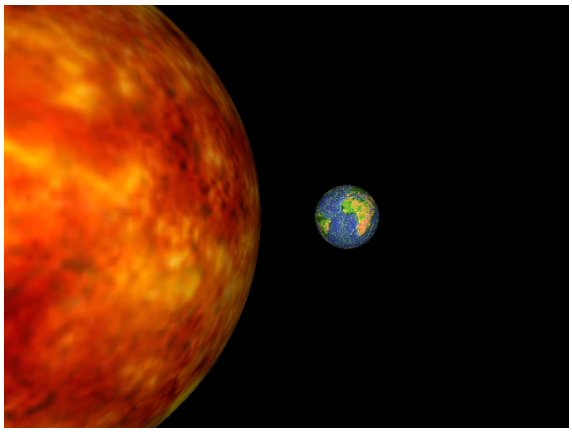

Abbildung 4: *Texturen auf Geometrien*

5.2 Freie Geometrien

Zwei Beispiele zu Geometrien

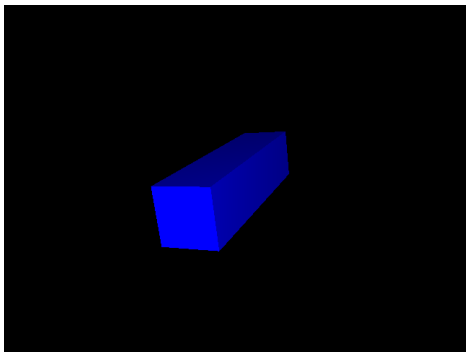

Abbildung 5: *final Node boxNode = new Node(new Transform().rotateY(-0.4).rotateX(-0.3).scale(1, 1, 4), new ArrayList<Geometry>()); boxNode.geos.add(new AxisAlignedBox(new LambertMaterial(new SingleColorTexture(new Color(0, 0, 1))))); geometries.add(boxNode);*

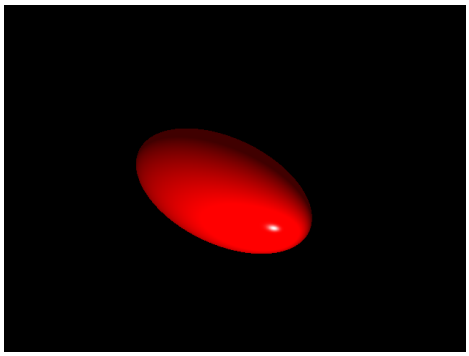

Abbildung 6: *final Node sphereNode = new Node(new Transform().rotateY(-0.9).rotateX(-0.3).scale(2.6, 1, 1.1), new ArrayList<Geometry>()); sphereNode.geos.add(new Sphere(new ReflectiveMaterial(new SingleColorTexture(new Color(1, 0, 0)), new SingleColorTexture(new Color(1, 1, 1)), new SingleColorTexture(new Color(0.8, 0.4, 0)), 64))); geometries.add(sphereNode);*