**Aufgabe 23**

1. $B_{i,m}(t)$ besitzt bei $t=\frac{i}{m}$ das Maximum.

Es ist zu zeigen: $\frac{d}{dt}B_{i,m}\left(\frac{i}{m}\right)=0$ und $\frac{d}{dt}\frac{d}{dt}B_{i,m}\left(\frac{i}{m}\right)<0$

Zuerst die Nullstelle:

$$B_{i,m}(t)=\binom{m}{i}\cdot t^i\cdot(1-t)^{m-i}$$

$$\frac{d}{dt}B_{i,m}(t)=\binom{m}{i}\cdot\left[i\cdot t^{i-1}\cdot(1-t)^{m-i}-t^i\cdot(m-i)\cdot(1-t)^{m-i-1}\right]$$

$$=\binom{m}{i}\cdot t^{i-1}\cdot(1-t)^{m-i-1}\cdot\left[i\cdot(1-t)-t\cdot(m-i)\right]$$

$$=\binom{m}{i}\cdot t^{i-1}\cdot(1-t)^{m-i-1}\cdot(i-t\cdot m)$$

$$\frac{d}{dt}B_{i,m}\left(\frac{i}{m}\right)=\binom{m}{i}\cdot\left(\frac{i}{m}\right)^{i-1}\cdot\left(1-\frac{i}{m}\right)^{m-i}\cdot\left[i-\frac{i}{m}\cdot m\right]$$

$$=\binom{m}{i}\cdot\left(\frac{i}{m}\right)^{i-1}\cdot\left(1-\frac{i}{m}\right)^{m-i}\cdot 0$$

$$=0$$

Und jetzt der Nachweis, dass es sich tatsächlich um ein Maximum handelt, indem ich die zweite Ableitung bilde:

$$\frac{d^2}{dt^2}B_{i,m}(t) = \frac{d}{dt}\left[\binom{m}{i}\cdot t^{i-1}\cdot(1-t)^{m-i-1}\cdot(i-t\cdot m)\right]$$

$$= \frac{d}{dt}\left[\binom{m}{i}\cdot t^{i-1}\cdot(1-t)^{m-i-1}\cdot i\right] - \frac{d}{dt}\left[\binom{m}{i}\cdot t^{i}\cdot(1-t)^{m-i-1}\cdot m\right]$$

$$= \binom{m}{i}\cdot i\cdot\frac{d}{dt}\left[t^{i-1}\cdot(1-t)^{m-i-1}\right] - \binom{m}{i}\cdot m\cdot\frac{d}{dt}\left[t^{i}\cdot(1-t)^{m-i-1}\right]$$

$$= \binom{m}{i}\cdot i\cdot\left[(i-1)\cdot t^{i-2}\cdot(1-t)^{m-i-1} + t^{i-1}\cdot(-m+i+1)\cdot(1-t)^{m-i-2}\right]$$

$$-\binom{m}{i}\cdot m\cdot\left[i\cdot t^{i-1}\cdot(1-t)^{m-i-1} + t^{i}\cdot(-m+i+1)\cdot(1-t)^{m-i-2}\right]$$

$$= \binom{m}{i}\cdot t^{i-2}\cdot(1-t)^{m-i-2}\cdot\left[i\cdot(i-1)\cdot(1-t) + t\cdot i\cdot(-m+i+1) - m\cdot i\cdot t\cdot(1-t) - m\cdot t^2\cdot(-m+i+1)\right]$$

$$= \binom{m}{i}\cdot t^{i-2}\cdot(1-t)^{m-i-2}\cdot\left[i\cdot(i-1)\cdot(1-t) + t\cdot i\cdot(-m+i+1) - m\cdot i\cdot t\cdot(1-t) - t^2\cdot(-m+i+1)\right]$$

$$= \binom{m}{i}\cdot t^{i-2}\cdot(1-t)^{m-i-2}\cdot\left[(1-t)\cdot\left[i\cdot(i-1) - m\cdot i\cdot t\right] + t\cdot(-m+i+1)\cdot(i-m\cdot t)\right]$$

$$= \binom{m}{i}\cdot\left(\frac{i}{m}\right)^{i-2}\cdot\left(1-\frac{i}{m}\right)^{m-i-2}\cdot\left[\left(1-\frac{i}{m}\right)\cdot\left[i\cdot(i-1) - m\cdot i\cdot\frac{i}{m}\right] + \frac{i}{m}\cdot(-m+i+1)\cdot\left(i-m\cdot\frac{i}{m}\right)\right]$$

$$= \binom{m}{i}\cdot\left(\frac{i}{m}\right)^{i-2}\cdot\left(1-\frac{i}{m}\right)^{m-i-2}\cdot\left[\left(1-\frac{i}{m}\right)\cdot\left[i^2-i-i^2\right] + \frac{i}{m}\cdot(-m+i+1)\cdot(i-i)\right]$$

$$= \binom{m}{i}\cdot\left(\frac{i}{m}\right)^{i-2}\cdot\left(1-\frac{i}{m}\right)^{m-i-2}\cdot\left[-i\cdot\left(1-\frac{i}{m}\right)\right]$$

$$= -\binom{m}{i}\cdot\left(\frac{i}{m}\right)^{i-2}\cdot\left(1-\frac{i}{m}\right)^{m-i-2}\cdot i\cdot\left(1-\frac{i}{m}\right)$$

Mit dem Wissen, dass $0 \le i \le m$ und daher $0 \le \frac{i}{m} \le 1$ gilt, stellt man fest, dass alle Faktor nicht-negativ sind. Lediglich das führende Minus wirkt sich auf das Vorzeichen aus. Da die zweite Ableitung somit stets nicht-positiv ist, handelt es sich um ein Maximum, für $i=0$ bzw. $i=m$ ist es eine Sattelstelle.

2. Summation: $\sum_{i=0}^{m} i \cdot B_{i,m}(t) = m \cdot t$

Eine von mir gerne benutzte Eigenschaft des Binomialkoeffizienten ist:

$$\binom{a}{b} = \frac{a!}{(a-b)! \cdot b!}$$

Sie lässt sich gewinnbringend einsetzen:

$$\begin{aligned}
\sum_{i=0}^{m} i \cdot B_{i,m}(t) &= \sum_{i=0}^{m} i \cdot \binom{m}{i} \cdot t^i \cdot (1-t)^{m-i} \\
&= \sum_{i=0}^{m} i \cdot \frac{m!}{(m-i)! \cdot i!} \cdot t^i \cdot (1-t)^{m-i} \\
&= \sum_{i=0}^{m} m \cdot t \cdot \frac{(m-1)!}{(m-i)! \cdot (i-1)!} \cdot t^{i-1} \cdot (1-t)^{m-i} \\
&= m \cdot t \cdot \sum_{i=0}^{m} \binom{m-1}{i-1} \cdot t^{i-1} \cdot (1-t)^{m-i} \\
&= m \cdot t \cdot \sum_{i=0}^{m} B_{i-1,m-1}(t) \\
&= m \cdot t
\end{aligned}$$

Im letzten Schritt nutze ich die Partition der 1 der Bernsteinpolynome.

3. Symmetrie: $B_{i,m}(t) = B_{m-i,m}(1-t)$

Die Herleitung ergibt sich durch konsequentes Einsetzen der Parameter, nachdem die Symmetrie der Binomialkoeffizienten verwendet wurde:

$$\binom{m}{m-i} = \frac{m!}{(m-[n-i])! \cdot (m-i)!} = \frac{m!}{(m-i)! \cdot i!} = \binom{m}{i}$$

$$\begin{aligned}
B_{m-i,m}(1-t) &= \binom{m}{m-i} \cdot (1-t)^{m-i} \cdot (1-[1-t])^{m-(m-i)} \\
&= \binom{m}{i} \cdot (1-t)^{m-i} \cdot t^i \\
&= B_{i,m}(t)
\end{aligned}$$

4. Differentiation:
Ich benutze in den folgenden Herleitungen jeweils die bereits in 1. entwickelte Gleichung:

$$\frac{d}{dt} B_{i,m}(t) = \binom{m}{i} \cdot t^{i-1} \cdot (1-t)^{m-i-1} \cdot (i - t \cdot m)$$

4a) Zuerst passe ich durch Brucherweiterung die Binomialkoeffizienten an, dann wird nur noch entsprechend zusammengefasst:

$$\begin{aligned}
& m \cdot \left(B_{i-1,m-1}(t) - B_{i,m-1}(t)\right) \\
&= m \cdot \left[ \binom{m-1}{i-1} \cdot t^{i-1} \cdot (1-t)^{m-i} - \binom{m-1}{i} \cdot t^{i} \cdot (1-t)^{m-i-1} \right] \\
&= m \cdot \left[ \binom{m}{i} \cdot \frac{i}{m} \cdot t^{i-1} \cdot (1-t)^{m-i-1} \cdot (1-t) - \binom{m}{i} \cdot \frac{m-i}{m} \cdot t^{i-1} \cdot t \cdot (1-t)^{m-i-1} \right] \\
&= \binom{m}{i} \cdot t^{i-1} \cdot (1-t)^{m-i-1} \cdot \left[ i \cdot (1-t) - t \cdot (m-i) \right] \\
&= \binom{m}{i} \cdot t^{i-1} \cdot (1-t)^{m-i-1} \cdot \left[ i - t \cdot m \right] \\
&= \frac{d}{dt} B_{i,m}(t)
\end{aligned}$$

4b) Allgemein gilt für Binomialkoeffizienten, dass $\binom{a}{0} = 1$ und daraus folgt dann:

$$\begin{aligned}
\frac{d}{dt} B_{0,m}(t) &= t^{-1} \cdot (1-t)^{m-1} \cdot (-t \cdot m) \\
&= -m \cdot (1-t)^{m-1} \\
B_{0,m-1}(t) &= \binom{m-1}{0} \cdot t^{0} \cdot (1-t)^{m-1} \\
&= (1-t)^{m-1} \\
\frac{d}{dt} B_{0,m}(t) &= -m \cdot (1-t)^{m-1} = -m \cdot B_{0,m-1}(t)
\end{aligned}$$

4c) Es ist stets $\binom{a}{a} = 1$:

$$\frac{d}{dt} B_{m,m}(t) = t^{m-1} \cdot (1-t)^{-1} \cdot (m - t \cdot m)$$

$$= t^{m-1} \cdot m$$

$$B_{m-1,m-1}(t) = \binom{m-1}{m-1} \cdot t^{m-1} \cdot (1-t)^0$$

$$= t^{m-1}$$

$$\frac{d}{dt} B_{m,m}(t) = t^{m-1} \cdot m = m \cdot B_{m-1,m-1}(t)$$

**Aufgabe 24**

Zuerst kläre ich die allgemeine Schreibweise einer Bèzierkurve zwischen den Punkten $P_a$, $P_{a+1}$, ..., $P_b$:

$$Q_{[a,b]}(t) = \sum_{i=a-1}^{b-1} \binom{b-a}{i-a+1} \cdot t^i \cdot (1-t)^{b-1-i} \cdot P_{i+1}$$

In dieser Art formuliere ich die Ausgangsfunktion:

$$Q_{[1,m+1]}(t) = \sum_{i=0}^{m} \binom{m}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1}$$

Meine Idee ist, dass die beiden Kurven niedrigeren Grades über die Punkte $P_2$ bis $P_m$ gemeinsam iterieren, lediglich $P_1$ und $P_{m+1}$ sind gesondert zu betrachten. Ich entferne sie daher aus der Summenformel und forme sie separat um.

$$\begin{aligned} Q_{[1,m+1]}(t) &= \binom{m}{0} \cdot t^0 \cdot (1-t)^m \cdot P_1 + \binom{m}{m} \cdot t^m \cdot (1-t)^{m-m} \cdot P_{m+1} + \sum_{i=1}^{m-1} \binom{m}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} \\ &= (1-t)^m \cdot P_1 + t^m \cdot P_{m+1} + \sum_{i=1}^{m-1} \binom{m}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} \end{aligned}$$

Als nächstes benutze ich eine Eigenschaft des Binomialkoeffizienten:

$$\binom{m}{i} = \binom{m-1}{i} + \binom{m-1}{i-1}$$

Somit kann man die Summenformel weiter zerlegen:

$$Q_{[1,m+1]}(t) = (1-t)^m \cdot P_1 + t^m \cdot P_{m+1} + \sum_{i=1}^{m-1} \binom{m-1}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} + \sum_{i=1}^{m-1} \binom{m-1}{i-1} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1}$$

Die anfänglich extrahierten Punkte kann man nun in die neuen Summenformeln einbauen:

$$\begin{aligned} (1-t)^m \cdot P_1 + \sum_{i=1}^{m-1} \binom{m-1}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} &= \sum_{i=0}^{m-1} \binom{m-1}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} \\ t^m \cdot P_{m+1} + \sum_{i=1}^{m-1} \binom{m-1}{i-1} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} &= \sum_{i=1}^{m} \binom{m-1}{i-1} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} \end{aligned}$$

Und man erhält als Ergebnis:

$$\begin{aligned} Q_{[1,m+1]}(t) &= \sum_{i=0}^{m-1} \binom{m-1}{i} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} + \sum_{i=1}^{m} \binom{m-1}{i-1} \cdot t^i \cdot (1-t)^{m-i} \cdot P_{i+1} \\ &= (1-t) \cdot \sum_{i=0}^{m-1} \binom{m-1}{i} \cdot t^i \cdot (1-t)^{m-i-1} \cdot P_{i+1} + t \cdot \sum_{i=1}^{m} \binom{m-1}{i-1} \cdot t^{i-1} \cdot (1-t)^{m-i} \cdot P_{i+1} \\ &= (1-t) \cdot Q_{[1,m]}(t) + t \cdot Q_{[2,m+1]}(t) \end{aligned}$$

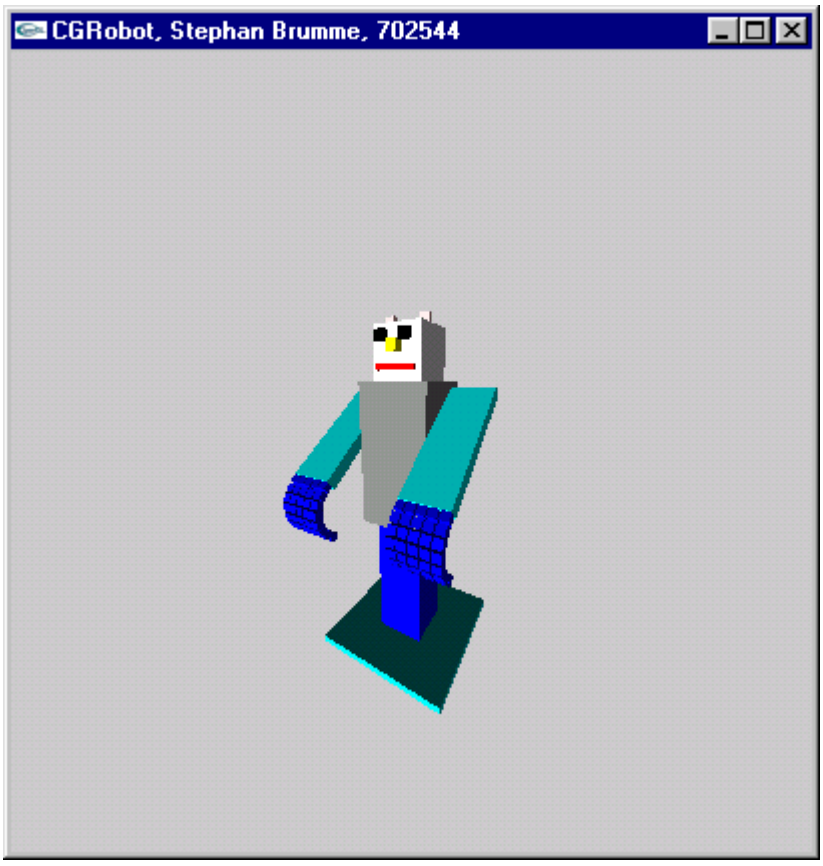
## Aufgabe 25

*Hinweis:* Ich habe den gefragten Quellcode für die Aufgaben 25 und 26 in einem Programm zusammengefasst. Deshalb gehe ich in meiner Dokumentation zu Aufgabe 26 nicht weiter auf den Programmrahmen oder die Bedienung ein.
Das Design des Szenengraphen und die Umsetzung im Quellcode ist in Zusammenarbeit mit Matthias Heise entstanden.

Der Roboter besitzt ein derart ausgeprägte Intelligenz, dass er nicht mehr manuell bedient werden muss. Die einzigen zur Verfügung stehenden Aktionen sind:

| Taste | Aktion |
|---|---|
| ESC | Programm beenden |
| Leertaste | Ablauf anhalten/fortfahren |
| c | Kameraflug Sinusbasiert/Bezierkurve |

Eine typische Situation gibt das folgende Bildschirmfoto wieder:




Alle Elemente des Roboters werden im Szenengraphen auf eine einzelne `Box` zurückgeführt, d.h. es existiert nur ein einzelner `LeafNode`. Ich habe mich sehr stark bemüht, dieses Ziel zu erreichen, um die Mächtigkeit dieses Designprinzips zu zeigen. Natürlich leidet die Performance etwas darunter, da eine allgemeine `Box` schon im Konstruktor gut positioniert und skaliert werden kann (geeignete Wahl der beiden Eckpunkte).
Bei den Attributen ging ich etwas großzügiger um, da die Unterschiede in Hinblick auf Winkel, Achsen usw. zu groß waren um sinnvolle Zusammenfassungen vorzunehmen.

Der entstandene Szenengraph ist recht groß (ich habe nicht genau nachgezählt, aber ich schätze, dass etwa 100 Objekte mit jeweils ca. 2 Attributen verwaltet werden). Auf der folgenden Seite befindet sich ein Grobskizze, die den wesentlichen Aufbau widerspiegelt.

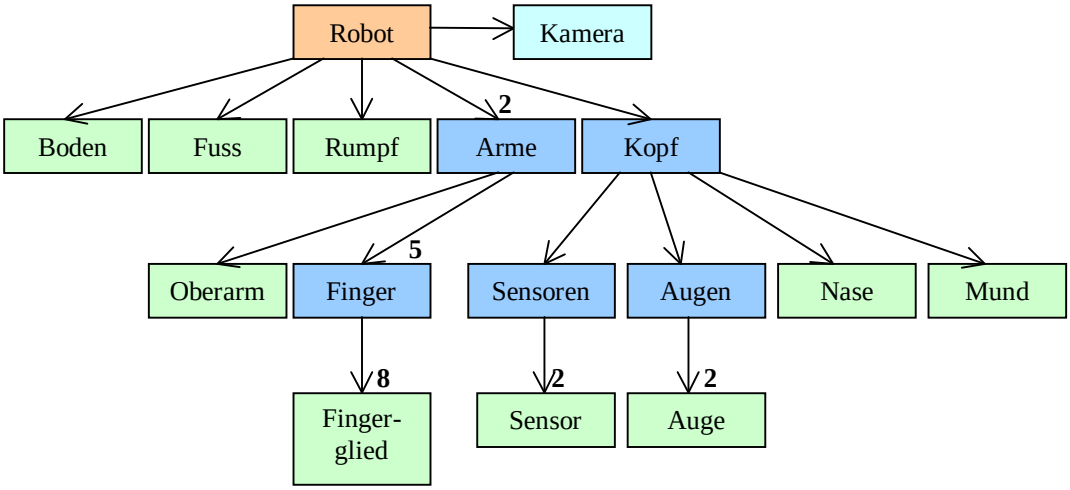



**Abbildung 1: Grobskizze Szenengraph**

Die Grafik stellt Blätter grün dar, sie bestehen nur aus einem `GroupNode`, der eine Referenz auf `LeafNode` (was überall die gleiche `Box` namens `leafBox` ist) und einige modifizierende Attribute enthält. Blaue Knoten sind ebenfalls vom Typ `GroupNode`, allerdings beinhalten sie nur Referenzen auf andere `GroupNode`s. Die Kamera hat eine Sonderstellung im Szenengraphen, deshalb ist sie auf oberster Ebene eingebunden worden. Die Zahlen quantifizieren die Anzahl der referenzierten Knoten, ich habe sie nur angegeben, wenn sie ungleich 1 ist.

Wie man aus der Grafik ersieht, ist es notwendig, dass ein Objekt mehrfach eingebunden werden kann (z.B. 5 Finger). Diese Fähigkeit erscheint mir derart essentiell, dass ich `GroupNode` um eine Klassenmethode `shareNode` erweiterte:

```
GroupNode* GroupNode::shareNode(Node* nodeObject, int nCount, Attribute* attrTransform1,
Attribute* attrTransform2)
{
    GroupNode* sharedNode = NULL;

    for (int shape = 1; shape <= nCount; shape++)
    {
        static GroupNode* oldNode;
        oldNode = sharedNode;
        sharedNode = new GroupNode;

        if (shape < nCount)
        {
            if (attrTransform1 != NULL)
                sharedNode->setAttribute(0, attrTransform1);
            if (attrTransform2 != NULL)
                sharedNode->setAttribute(1, attrTransform2);
        }

        sharedNode->setChildNode(0, nodeObject);

        if (oldNode != NULL)
            sharedNode->setChildNode(1, oldNode);
    }

    return sharedNode;
}
```

Die Kernfunktionalität der mehrfachen Benutzung eines Objektes wird erheblich erschwert, da ich verschachtelte Attribute haben will, d.h. ein eher sequentieller Charakter statt eines parallelen benötige. Um den Grund zu verdeutlichen, ziehe ich wieder das Beispiel mit den Fingern einer Hand zur Hilfe: Die Finger sollen nicht an der gleichen Position sich befinden, sondern jeweils verschoben werden. Eine Translation des dritten Fingers soll aber auch indirekt den ersten beeinflussen, damit er sich nicht mit dem zweiten überlappt.

Grafisch veranschaulicht sieht man den Unterschied. Der naive Ansatz ist falsch, da die Attribute nur jeweils auf einen Finger wirken:

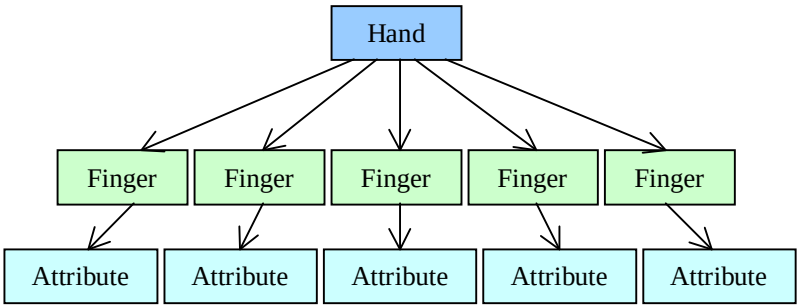



**Abbildung 2: Naives shareNode**

Richtig dagegen ist:

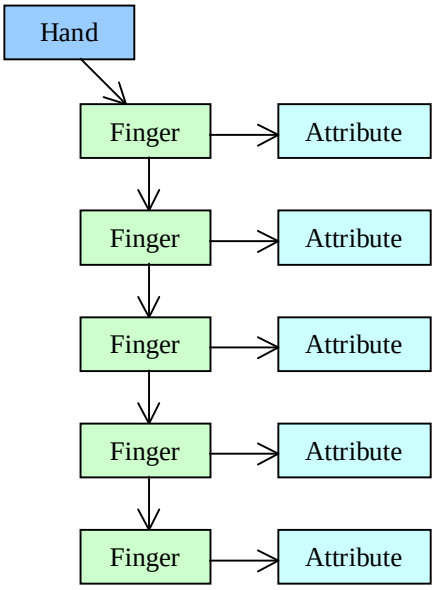



**Abbildung 3: Korrektes shareNode**

In beiden Zeichnungen wird eventuell nicht ganz ersichtlich, dass effektiv nur 1 Finger und 1 Attribut existieren. Allerdings werden 5 `GroupNode`s erzeugt, die jeweils auf den gleichen Finger bzw. das Attribut verweisen.

Die Umsetzung der Transformationen `Rotation` und `Scaling` gestaltete sich sehr einfach, da sie vom Grundmuster her den gleichen Aufbau wie `Translation` haben.

```
//
// Scaling
//

Scaling::Scaling(const Vector& scale)
    : scale_(scale) {
}

const Vector& Scaling::getScaling() const {
    return scale_;
}

void Scaling::setScaling(const Vector& scale) {
    scale_ = scale;
}

void Scaling::set() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPushMatrix(); // Speichere die aktuelle Matrix
    glScaled(scale_[0], scale_[1], scale_[2]); // Skaliere
}

void Scaling::unset() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPopMatrix(); // Restauriere die vorherige Matrix
}

//
// Rotation
//

Rotation::Rotation(double angle, const Vector& axis)
    : axis_(axis) {
    setAngle(angle);
}

double Rotation::getAngle() const {
    return angle_;
}

const Vector& Rotation::getAxis() const {
    return axis_;
}

void Rotation::setAngle(double angle) {
    angle_ = angle;
    while(angle_ >= 360) { angle_ -= 360; }
    while(angle_ <   0) { angle_ += 360; }
}

void Rotation::setAxis(const Vector& axis) {
    axis_ = axis;
}

void Rotation::set() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPushMatrix(); // Speichere die aktuelle Matrix
    glRotated(angle_, axis_[0], axis_[1], axis_[2]); // Rotiere
}

void Rotation::unset() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPopMatrix(); // Restauriere die vorherige Matrix
}
```

Ich musste zusätzlich die Methode `Rotation::setAngle` korrigieren, da der alte Code falsch war und ungültige Werte zuließ (außerhalb des Bereiches 0°...359°).

Beim Entwurf der `render`-Methode der Klasse `Box` trieb mich der Gedanke, dass ich möglichst keinen direkten OpenGL-Zugriff haben sollte, um so einen gewissen Abstraktionsgrad zu wahren. Als Resultat baue ich den Quader aus 6*2 Dreiecken zusammen. Unter Beachtung des Gegen-Uhrzeigersinns bei der Generierung erhalte ich die jeweiligen Flächennormalen als integralen Bestandteil von `Triangle` quasi gratis:

```
void Box::render() {
    // damit das Tippen leichter fällt ...
#define l lowerLeftFront_
#define r upperRightBack_

    // oben
    Triangle oben1(Vector(l[0], r[1], l[2]),
                   Vector(r[0], r[1], l[2]),
                   Vector(r[0], r[1], r[2]));
    Triangle oben2(Vector(l[0], r[1], l[2]),
                   Vector(r[0], r[1], r[2]),
                   Vector(l[0], r[1], r[2]));
    oben1.render();
    oben2.render();

    // unten
    Triangle unten1(Vector(l[0], l[1], l[2]),
                    Vector(r[0], l[1], l[2]),
                    Vector(r[0], l[1], r[2]));
    Triangle unten2(Vector(l[0], l[1], l[2]),
                    Vector(r[0], l[1], r[2]),
                    Vector(l[0], l[1], r[2]));
    unten1.render();
    unten2.render();

    // hinten
    Triangle hinten1(Vector(l[0], l[1], r[2]),
                     Vector(r[0], l[1], r[2]),
                     Vector(r[0], r[1], r[2]));
    Triangle hinten2(Vector(l[0], l[1], r[2]),
                     Vector(r[0], r[1], r[2]),
                     Vector(l[0], r[1], r[2]));
    hinten1.render();
    hinten2.render();

    // vorn
    Triangle vorn1(Vector(l[0], l[1], l[2]),
                   Vector(r[0], l[1], l[2]),
                   Vector(r[0], r[1], l[2]));
    Triangle vorn2(Vector(l[0], l[1], l[2]),
                   Vector(r[0], r[1], l[2]),
                   Vector(l[0], r[1], l[2]));
    vorn1.render();
    vorn2.render();

    // links
    Triangle links1(Vector(l[0], l[1], l[2]),
                    Vector(l[0], r[1], r[2]),
                    Vector(l[0], l[1], r[2]));
    Triangle links2(Vector(l[0], l[1], l[2]),
                    Vector(l[0], r[1], l[2]),
                    Vector(l[0], r[1], r[2]));
    links1.render();
    links2.render();

    // rechts
    Triangle rechts1(Vector(r[0], l[1], l[2]),
                     Vector(r[0], r[1], r[2]),
                     Vector(r[0], l[1], r[2]));
    Triangle rechts2(Vector(r[0], l[1], l[2]),
                     Vector(r[0], r[1], l[2]),
                     Vector(r[0], r[1], r[2]));
    rechts1.render();
    rechts2.render();
#undef l
#undef r
}
```

## Aufgabe 26

*Hinweis:* Aufgrund eines Versehens meinerseits heißt die Kameraklasse `Camera` und nicht – wie in der Aufgabenstellung gefordert – `lookAt`.

Die Klasse `Camera` leitet von `Transformation` ab und besitzt zusätzlich die Attribute `from_`, `to_` und `up_` welche als `Vector` die Position und Ausrichtung der Kamera erfassen. Entsprechend dem Kapselungsgedanken ist der direkte Zugriff auf diese Variablen nicht erlaubt, sondern erfolgt über passende Methoden:

```
//
// Camera
//

Camera::Camera(const Vector& from, const Vector& to, const Vector& up)
       : from_(from), to_(to), up_(up)
{
}

void Camera::set()
{
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    gluLookAt(from_[0], from_[1], from_[2],
              to_[0],   to_[1],   to_[2],
              up_[0],   up_[1],   up_[2]);
}

void Camera::unset()
{
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
}

void Camera::setFrom(const Vector& from)
{
    from_ = from;
}

void Camera::setTo(const Vector& to)
{
    to_ = to;
}

void Camera::setUp(const Vector& up)
{
    up_ = up;
}

Vector Camera::getFrom() const
{
    return from_;
}

Vector Camera::getTo() const
{
    return to_;
}

Vector Camera::getUp() const
{
    return up_;
}
```

In Aufgabe 25 habe ich schon kurz angerissen, dass die Kamera ein Attribut des Wurzelknotens ist. Ich verweise deshalb auf Abbildung 1, die den ganzen Zusammenhang noch grafisch veranschaulicht.

In der Methode `CGRobot::onTimer` erfolgt die Veränderung der Kameraposition. Ich habe zwei Bahnen vordefiniert: standardmäßig wird eine Bezierkurve verfolgt, mit der Taste „`c`“ kann man auf eine einfachere Kurve umschalten, die sich aus `sin`- und `cos`-Beziehungen ergibt:

```
void CGRobot::onTimer() {
    if (!constructed_)
        return;

    static int nTimerTicks = 0;

    // durchgängiger Timer
    nTimerTicks++;

    // Bewegung
    if(!pause_)
    {
        // Kamera bewegen
        if (cameraBezier_)
        {
            // auf Bezierkurve
            const int bezierPoints = 4;
            Vector v[bezierPoints] = { Vector(-2,-2, 3),
                                       Vector( 5, 2, 7),
                                       Vector( 3, 5,-5),
                                       Vector(-2,-2,-2) };

            static double t = 0.01;
            static double incT = 0.01;

            if (t<0.01 || t>0.99)
                incT = -incT;
            t += incT;
            camera_->setFrom(bezier(v,bezierPoints,t));
        }
        else
        {
            // auf vordefinierter Bahn
            double angle = nTimerTicks*3.1415926/180.0;
            camera_->setFrom(Vector(-3*sin(angle), 3*sin(angle), 3*cos(angle)));
        }

        // Finger öffnen/schliessen
        static double fingerAngleInc = -1;
        double angle = fingerRotation_->getAngle();
        if (angle == 340 || angle == 0)
            fingerAngleInc = -fingerAngleInc;
        fingerRotation_->setAngle(angle+fingerAngleInc);

        // Arme bewegen
        static double armAngleInc = -1;
        angle = armRotation_->getAngle();
        if (angle == 80 || angle == 30)
            armAngleInc = -armAngleInc;
        armRotation_->setAngle(angle+armAngleInc);
    }

    // blinken
    delete sensorColor_;
    if (nTimerTicks % 2 == 0)
        sensorColor_ = new Color(1,0,0);
    else
        sensorColor_ = new Color(1,0.8,0.8);

    needsRedraw();
}
```

Das Grundprinzip beruht darauf, dass im Szenengraphen die Variable `camera_` existiert, auf die ich von außen zugreifen kann. Es reicht dabei aus, nur den `Look`-Vektor zu manipulieren. Als kleine Gimmicks erlaube ich, dass der Roboter Arme und Finger bewegen kann sowie dass die Sensoren blinken.

Die Bezierkurve wird mit dem Algorithmus von de Casteljau berechnet. Ich empfand es als günstig, diese Routine im Modul von `Vector` als `friend` unterzubringen und so eine Ähnlichkeit zu Operationen wie das Skalarprodukt herzustellen:

```
inline Vector bezier(const Vector* v, int nPoints, double t)
{
    Vector pts[10];

    for (int n=0; n<nPoints; n++)
        pts[n] = v[n];

    for(int outer=1; outer< nPoints; outer++)
        for(int inner=0; inner<= (nPoints-outer); inner++)
            pts[inner]+= t* (pts[inner+1]-pts[inner]);

    return pts[0];
}
```

## Quellcode

Ich nahm in allen Klassen Erweiterungen oder Korrekturen vor, die ich weiter oben schon im wesentlichen besprochen habe. Aus diesem Grunde ist dieser Anhang recht umfangreich und ich empfehle eher das Laden des Quellcodes im Visual Studio.

*Vector.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef _VECTOR_H
#define _VECTOR_H

#include <iostream.h>
#include <math.h>

//- Vector
class Vector {
//. 3D Vector Class.
public:
    //- Vector
    Vector(double x = 0, double y = 0, double z = 0);

    //- operator==, operator!=, operator<
    bool operator==(const Vector&) const;
    bool operator!=(const Vector&) const;
    bool operator<(const Vector&) const;
        //. The comparison is based on the lexiographic order.

    //- operator[]
    double& operator[](int i);
    double operator[](int i) const;
        //. Indices 0,1, and 2 correspond to the x,y,z coordinates
        //. of the vector.
        //. CAUTION: For performance reasons, indices are not checked.

    //- operator+=, operator-=, operator*=
    Vector& operator+=(const Vector&);
    Vector& operator-=(const Vector&);
    Vector& operator*=(const Vector&);
    Vector& operator*=(double);

    //- operator+, operator-, operator*, operator/, operator-, operator*
    Vector operator+(const Vector&) const;
    Vector operator-(const Vector&) const;
    Vector operator*(const Vector&) const;
    Vector operator/(double) const;
    Vector operator-() const;
    Vector operator*(double) const;
    friend inline Vector operator*(double, const Vector&);

    //- normalized, normalize
    Vector normalized() const;
    bool normalize();
        //. `normalized' returns a normalized copy of the vector object.
        //. `normalize' normalizes the vector object. If the length
        //. of the vector was > 0, it returns 1 for success, otherwise 0.

    //- abs, abs2, dotProduct
    friend inline double abs(const Vector&);
    friend inline double abs2(const Vector&);
    friend inline double dotProduct(const Vector&, const Vector&);
        //. Vector length and scalar product.

    //- generate a point on a Bezier curve, max degree of 10
    friend inline Vector bezier(const Vector* v, int nPoints, double t);
```

```
    //- makeCWTriangleNormal, makeCCWTriangleNormal
    friend inline Vector makeCWTriangleNormal (
        const Vector& p0, const Vector& p1, const Vector& p2
    );
    friend inline Vector makeCCWTriangleNormal (
        const Vector& p0, const Vector& p1, const Vector& p2
    );
        //. Computes ((p0-p1)*(p1-p2)).normalized, i.e. a vector that is
        //. perpendicular the the plane defined by the three vertices.

    //- operator<<, operator>>
    friend  ostream& operator<<(ostream&, const Vector&);
    friend  istream& operator>>(istream&, Vector&);

    //- rep
    double* rep() const;
        //. Direct data access. It is guaranteed that the
        //. vector components are stored continuously.


private:
    double v_[3];
};

inline Vector::Vector(double x, double y, double z) {
    v_[0] = x; v_[1] = y; v_[2] = z;
}

inline double Vector::operator[](int i) const { return v_[i]; }

inline double& Vector::operator[](int i) { return v_[i]; }

inline bool Vector::operator==(const Vector& u) const {
    return(v_[0]==u.v_[0] && v_[1]==u.v_[1] && v_[2]==u.v_[2]);
}

inline bool Vector::operator!=(const Vector& u) const {
    return(v_[0]!=u.v_[0] || v_[1]!=u.v_[1] || v_[2]!=u.v_[2]);
}

inline Vector& Vector::operator+=(const Vector& u) {
    v_[0] += u.v_[0];
    v_[1] += u.v_[1];
    v_[2] += u.v_[2];
    return *this;
}

inline Vector& Vector::operator-=(const Vector& u) {
    v_[0] -= u.v_[0];
    v_[1] -= u.v_[1];
    v_[2] -= u.v_[2];
    return *this;
}

inline Vector& Vector::operator*=(double t) {
    v_[0] *= t;
    v_[1] *= t;
    v_[2] *= t;
    return *this;
}

inline Vector Vector::operator+(const Vector& u) const {
    return Vector(v_[0]+u.v_[0],v_[1]+u.v_[1],v_[2]+u.v_[2]);
}

inline Vector Vector::operator-(const Vector& u) const {
    return Vector(v_[0]-u.v_[0],v_[1]-u.v_[1],v_[2]-u.v_[2]);
}

inline Vector Vector::operator*(const Vector& w) const {
    return Vector(v_[1]*w.v_[2]-v_[2]*w.v_[1],
                  v_[2]*w.v_[0]-v_[0]*w.v_[2],
                  v_[0]*w.v_[1]-v_[1]*w.v_[0]);
}
```

```
inline Vector Vector::operator/(double s) const {
    return Vector(v_[0]/s, v_[1]/s, v_[2]/s);
}

inline Vector Vector::operator-() const {
    return Vector(-v_[0], -v_[1], -v_[2]);
}

inline Vector Vector::operator*(double f) const {
    return Vector(v_[0]*f, v_[1]*f, v_[2]*f);
}

inline Vector operator*(double s, const Vector& u) {
    return Vector(u[0]*s, u[1]*s, u[2]*s);
}

inline double dotProduct(const Vector& u1, const Vector& u2) {
    return(u1[0]*u2[0] + u1[1]*u2[1] + u1[2]*u2[2]);
}

inline double abs(const Vector& u) { return(sqrt(dotProduct(u,u))); }
inline double abs2(const Vector& u) { return dotProduct(u,u); }

inline Vector bezier(const Vector* v, int nPoints, double t)
{
    Vector pts[10];

    for (int n=0; n<nPoints; n++)
        pts[n] = v[n];

    for(int outer=1; outer< nPoints; outer++)
        for(int inner=0; inner<= (nPoints-outer); inner++)
            pts[inner]+= t* (pts[inner+1]-pts[inner]);

    return pts[0];
}

inline Vector makeCWTriangleNormal(const Vector& p0, const Vector& p1, const Vector& p2) {
    return(((p0 - p1) * (p1 - p2)).normalized());
}

inline Vector makeCCWTriangleNormal(const Vector& p0, const Vector& p1, const Vector& p2) {
    return(makeCWTriangleNormal(p2, p1, p0));
}

inline double* Vector::rep() const {
    return (double*)(&(v_[0]));
}

#endif // _VECTOR_H
```

*Vector.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "vector.h"
#include <math.h>
#include <string.h>

bool Vector::operator<(const Vector& u) const {
    return((v_[0]<u.v_[0]) ||
           (v_[0]==u.v_[0] && v_[1]<u.v_[1]) ||
           (v_[0]==u.v_[0] && v_[1]==u.v_[1] && v_[2]<u.v_[2])
    );
}

Vector& Vector::operator*=(const Vector& u) { // cross product
    double x = v_[0];
    double y = v_[1];
    double z = v_[2];
    v_[0] = y*u.v_[2] - z*u.v_[1];
    v_[1] = z*u.v_[0] - x*u.v_[2];
    v_[2] = x*u.v_[1] - y*u.v_[0];
    return *this;
}

Vector Vector::normalized() const {
    double L = sqrt(v_[0]*v_[0] + v_[1]*v_[1] + v_[2]*v_[2]);
    if(fabs(L) == 0.0) {
        return Vector(0,0,0);
    } else {
        return Vector(v_[0]/L, v_[1]/L, v_[2]/L);
    }
}

bool Vector::normalize() {
    double L = sqrt(v_[0]*v_[0] + v_[1]*v_[1] + v_[2]*v_[2]);
    if(fabs(L) == 0.0) {
        return false;
    } else {
        v_[0] /= L;
        v_[1] /= L;
        v_[2] /= L;
        return true;
    }
}

//
// IO
//

ostream& operator<<(ostream& s, const Vector& u) {
    s << u[0] << " " << u[1] << " " << u[2];
    return s;
}

istream& operator>>(istream& s, Vector& u) {

    s >> u[0];
    if(!s.good()) { return s; }

    s >> u[1];
    if(!s.good()) { return s; }

    s >> u[2];
    if(!s.good()) { return s; }

    return s;
}
```

*Node.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef NODE_H
#define NODE_H

#include "attribute.h"
#include "shape.h"

//
// Konstanten
//

// Maximale Anzahl an Attributen pro GroupNode
static const unsigned int MAX_ATTRIBUTES  = 20;

// Maximale Anzahl an Kind-Knoten pro GroupNode
static const unsigned int MAX_CHILD_NODES = 20;

//
// Node
//

class Node {
public:
    // Wird zur Traversierung des Szenengraphen
    // aufgerufen.
    virtual void traverse() = 0;
};

//
// LeafNode
//

class LeafNode : public Node {
public:
    LeafNode(Shape* shape);
    // Jeder Blatt-Knoten enthaelt genau ein Shape,
    // welches bei der Traversierung des Szenengraphen
    // gerendert wird (Shape::render()).

    virtual void traverse();

private:
    Shape* shape_;
};

//
// GroupNode
//

class GroupNode : public Node {
public:
    GroupNode();

    static GroupNode* GroupNode::shareNode(Node* nodeObject, int nCount,
                     Attribute* attrTransform1 = NULL, Attribute* attrTransform2 = NULL);

    // Jeder Gruppen-Knoten kann maximal MAX_ATTRIBUTES-viele
    // Attribute und MAX_CHILD_NODES-viele Kind-Knoten aufnehmen.

    // Mit diesen Methoden koennen die Attribute ausgelesen und
    // gesetzt werden. Um ein Attribut zu entfernen schreibt man
    // an die entsprechende Stelle einen NULL-Zeiger.
    virtual Attribute* getAttribute(unsigned int index) const;
    virtual void setAttribute(unsigned int index, Attribute* attribute);

    // Mit diesen Methoden koennen die Kind-Knoten ausgelesen und
    // gesetzt werden. Um einen Kind-Knoten zu entfernen schreibt man
    // an die entsprechende Stelle einen NULL-Zeiger.
```

```
    virtual Node* getChildNode(unsigned int index) const;
    virtual void setChildNode(unsigned int index, Node* child);

    // Wird ein Gruppen-Knoten traversiert, so werden zuerst alle
    // enthaltenen Attribute in der angegebenen Reihenfolge gesetzt
    // (Attribute::set()). Dann werden alle Kind-Knoten traversiert
    // (Node::traverse()). Und anschliessend werden alle Attribute
    // wieder in umgekehrter Reihenfolge zurueckgenommen
    // (Attribute::unset()).
    virtual void traverse();

private:
    Attribute* attributes_[MAX_ATTRIBUTES];
    Node*      children_[MAX_CHILD_NODES];
};

#endif // NODE_H
```

*Node.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "node.h"

#include <assert.h>
#include <iostream.h>
#include <GL/glut.h>

//
// LeafNode
//

LeafNode::LeafNode(Shape* shape)
    : shape_(shape) {
}

void LeafNode::traverse() {
    if(shape_) { shape_->render(); }

//  cout << shape_ << "      ";
}

//
// GroupNode
//

GroupNode::GroupNode() {
    // Initialisiere alle Attribute mit NULL
    for(unsigned int i = 0; i < MAX_ATTRIBUTES; ++i) {
        attributes_[i] = 0;
    }

    // Initialisiere alle Kind-Knoten mit NULL
    for(unsigned int j = 0; j < MAX_CHILD_NODES; ++j) {
        children_[j] = 0;
    }
}

GroupNode* GroupNode::shareNode(Node* nodeObject, int nCount, Attribute* attrTransform1,
Attribute* attrTransform2)
{
    GroupNode* sharedNode = NULL;

    for (int shape = 1; shape <= nCount; shape++)
    {
        static GroupNode* oldNode;
        oldNode = sharedNode;
        sharedNode = new GroupNode;

        if (shape < nCount)
        {
            if (attrTransform1 != NULL)
                sharedNode->setAttribute(0, attrTransform1);
            if (attrTransform2 != NULL)
                sharedNode->setAttribute(1, attrTransform2);
        }

        sharedNode->setChildNode(0, nodeObject);

        if (oldNode != NULL)
            sharedNode->setChildNode(1, oldNode);
    }

    return sharedNode;
}


Attribute* GroupNode::getAttribute(unsigned int index) const {
    assert(index < MAX_ATTRIBUTES);
```

```
    return attributes_[index];
}

void GroupNode::setAttribute(unsigned int index, Attribute* attribute) {
    assert(index < MAX_ATTRIBUTES);
    attributes_[index] = attribute;
}

Node* GroupNode::getChildNode(unsigned int index) const {
    assert(index < MAX_CHILD_NODES);
    return  children_[index];
}

void GroupNode::setChildNode(unsigned int index, Node* child) {
    assert(index < MAX_CHILD_NODES);
    children_[index] = child;
}

void GroupNode::traverse() {
    // Setzte alle Attribute
    for(unsigned int i = 0; i < MAX_ATTRIBUTES; ++i) {
        if(attributes_[i]) { attributes_[i]->set(); }
    }

    // Traversiere alle Kind-Knoten
    for(unsigned int j = 0; j < MAX_CHILD_NODES; ++j) {
        if(children_[j]) { children_[j]->traverse(); }
    }

    // Setzte alle Attribute in umgekehrter Reihenfolge zurueck
    for(unsigned int k = MAX_ATTRIBUTES; k > 0; --k) {
        if(attributes_[k - 1]) { attributes_[k - 1]->unset(); }
    }
}
```

*Shape.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef SHAPE_H
#define SHAPE_H

#include "vector.h"

//
// Shape
//

class Shape {
public:
    // Jedes Shape kann sich selbst mit Hilfe von
    // OpenGL-Befehlen rendern.
    virtual void render() = 0;
};

//
// Triangle
//

class Triangle : public Shape {
public:
    Triangle(const Vector& p0, const Vector& p1, const Vector& p2);
    // Zeichnet das angegebene Dreieck. Die Normale zum Dreieck wird
    // automatisch berechnet. Das Dreieck muss im Gegenuhrzeigersinn
    // spezifiziert werden.

    virtual void render();

private:
    Vector points_[3];
};

//
// Box
//

class Box : public Shape {
public:
    Box(const Vector& lowerLeftFront, const Vector& upperRightBack);
    // Zeichnet den angegebenen Quader.

    virtual void render();

private:
    Vector lowerLeftFront_;
    Vector upperRightBack_;
};

#endif // SHAPE_H
```

*Shape.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "shape.h"

#include <iostream.h>
#include <GL/glut.h>

//
// Hilfsfunktionen
//

static inline double min(double a, double b) {
    return((a < b) ? a : b);
}

static inline double max(double a, double b) {
    return((a > b) ? a : b);
}

// Berechnet die Normale zum Dreieck. Die Dreieckseckpunkte muessen
// dazu im Gegenuhrzeigersinn angegeben werden.
static inline Vector normal(const Vector& p0, const Vector& p1, const Vector& p2) {
    // Verwende die ueberladenen Operatoren von Vector
    return((p1 - p0) * (p0 - p2));
}

//
// Triangle
//

Triangle::Triangle(const Vector& p0, const Vector& p1, const Vector& p2) {
    points_[0] = p0;
    points_[1] = p1;
    points_[2] = p2;
}

void Triangle::render() {
    glBegin(GL_TRIANGLES);
    glNormal3dv(normal(points_[0], points_[1], points_[2]).rep());
    glVertex3dv(points_[0].rep());
    glVertex3dv(points_[1].rep());
    glVertex3dv(points_[2].rep());
    glEnd();
}

//
// Box
//

Box::Box(const Vector& llf, const Vector& urb) {
    lowerLeftFront_ = Vector(min(llf[0], urb[0]),
                             min(llf[1], urb[1]),
                             min(llf[2], urb[2]));

    upperRightBack_ = Vector(max(llf[0], urb[0]),
                             max(llf[1], urb[1]),
                             max(llf[2], urb[2]));
}

void Box::render() {
    // damit das Tippen leichter fällt ...
#define l lowerLeftFront_
#define r upperRightBack_

    // oben
    Triangle oben1(Vector(l[0], r[1], l[2]),
                   Vector(r[0], r[1], l[2]),
                   Vector(r[0], r[1], r[2]));
    Triangle oben2(Vector(l[0], r[1], l[2]),
```

```
                Vector(r[0], r[1], r[2]),
                Vector(l[0], r[1], r[2]));
    oben1.render();
    oben2.render();

    // unten
    Triangle unten1(Vector(l[0], l[1], l[2]),
                    Vector(r[0], l[1], l[2]),
                    Vector(r[0], l[1], r[2]));
    Triangle unten2(Vector(l[0], l[1], l[2]),
                    Vector(r[0], l[1], r[2]),
                    Vector(l[0], l[1], r[2]));
    unten1.render();
    unten2.render();

    // hinten
    Triangle hinten1(Vector(l[0], l[1], r[2]),
                     Vector(r[0], l[1], r[2]),
                     Vector(r[0], r[1], r[2]));
    Triangle hinten2(Vector(l[0], l[1], r[2]),
                     Vector(r[0], r[1], r[2]),
                     Vector(l[0], r[1], r[2]));
    hinten1.render();
    hinten2.render();

    // vorn
    Triangle vorn1(Vector(l[0], l[1], l[2]),
                   Vector(r[0], l[1], l[2]),
                   Vector(r[0], r[1], l[2]));
    Triangle vorn2(Vector(l[0], l[1], l[2]),
                   Vector(r[0], r[1], l[2]),
                   Vector(l[0], r[1], l[2]));
    vorn1.render();
    vorn2.render();

    // links
    Triangle links1(Vector(l[0], l[1], l[2]),
                    Vector(l[0], r[1], r[2]),
                    Vector(l[0], l[1], r[2]));
    Triangle links2(Vector(l[0], l[1], l[2]),
                    Vector(l[0], r[1], l[2]),
                    Vector(l[0], r[1], r[2]));
    links1.render();
    links2.render();

    // rechts
    Triangle rechts1(Vector(r[0], l[1], l[2]),
                     Vector(r[0], r[1], r[2]),
                     Vector(r[0], l[1], r[2]));
    Triangle rechts2(Vector(r[0], l[1], l[2]),
                     Vector(r[0], r[1], l[2]),
                     Vector(r[0], r[1], r[2]));
    rechts1.render();
    rechts2.render();
#undef l
#undef r
}
```

*Transformation.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef TRANSFORMATION_H
#define TRANSFORMATION_H

#include "attribute.h"
#include "Vector.h"

//
// Transformation
//

class Transformation : public Attribute {
public:
    // Transformationen sind spezielle Attribute.

    virtual void set() = 0;
    virtual void unset() = 0;
};

//
// Camera
//

class Camera : public Transformation {
public:
    Camera(const Vector& from = Vector(0,0,3),
           const Vector& to   = Vector(0,0,0),
           const Vector& up   = Vector(0,1,0));
    // erzeugt eine Kamera

    virtual void set();
    virtual void unset();

    // Zugriff auf Kameraeigenschaften
    Vector getFrom() const;
    Vector getTo  () const;
    Vector getUp  () const;
    void   setFrom(const Vector& from);
    void   setTo  (const Vector& to);
    void   setUp  (const Vector& up);

private:
    Vector from_, to_, up_;
};

//
// Translation
//

class Translation : public Transformation {
public:
    Translation(const Vector& trans = Vector(0, 0, 0));
    // Verschiebt das Koordinatensystem um den Vektor trans.

    // Mit diesen Methoden kann man die Translation
    // auslesen und setzen.
    const Vector& getTranslation() const;
    void setTranslation(const Vector& trans);

    virtual void set();
    virtual void unset();

private:
    Vector trans_;
};

//
// Scaling
```

```
//

class Scaling : public Transformation {
public:
    Scaling(const Vector& scale = Vector(1, 1, 1));
    // Skaliert das Koordinatensystem um die in scale
    // angegebenen Faktoren.

    // Mit diesen Methoden kann man die Skalierung
    // auslesen und setzen.
    const Vector& getScaling() const;
    void setScaling(const Vector& scale);

    virtual void set();
    virtual void unset();

private:
    Vector scale_;
};

//
// Rotation
//

class Rotation : public Transformation {
public:
    Rotation(double angle, const Vector& axis = Vector(0, 0, 1));
    // Rotiert das Koordinatensystem um angle Grad um die
    // angegebene Rotationsachse.

    // Mit diesen Methoden kann man den Rotationswinkel
    // auslesen und setzen.
    double getAngle() const;
    void setAngle(double angle);

    // Mit diesen Methoden kann man die Rotationsachse
    // auslesen und setzen.
    const Vector& getAxis() const;
    void setAxis(const Vector& axis);

    virtual void set();
    virtual void unset();

private:
    double angle_;
    Vector axis_;
};

#endif // TRANSFORMATION_H
```

*Transformation.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "transformation.h"

#include <iostream.h>
#include <GL/glut.h>

//
// Camera
//

Camera::Camera(const Vector& from, const Vector& to, const Vector& up)
       : from_(from), to_(to), up_(up)
{
}

void Camera::set()
{
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    gluLookAt(from_[0], from_[1], from_[2],
              to_[0],   to_[1],   to_[2],
              up_[0],   up_[1],   up_[2]);
}

void Camera::unset()
{
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
}

void Camera::setFrom(const Vector& from)
{
    from_ = from;
}

void Camera::setTo(const Vector& to)
{
    to_ = to;
}

void Camera::setUp(const Vector& up)
{
    up_ = up;
}

Vector Camera::getFrom() const
{
    return from_;
}

Vector Camera::getTo() const
{
    return to_;
}

Vector Camera::getUp() const
{
    return up_;
}


//
// Translation
//

Translation::Translation(const Vector& trans)
    : trans_(trans) {
```

```
}

const Vector& Translation::getTranslation() const {
    return trans_;
}

void Translation::setTranslation(const Vector& trans) {
    trans_ = trans;
}

void Translation::set() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPushMatrix(); // Speichere die aktuelle Matrix
    glTranslated(trans_[0], trans_[1], trans_[2]); // Transliere
}

void Translation::unset() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPopMatrix(); // Restauriere die vorherige Matrix
}

//
// Scaling
//

Scaling::Scaling(const Vector& scale)
    : scale_(scale) {
}

const Vector& Scaling::getScaling() const {
    return scale_;
}

void Scaling::setScaling(const Vector& scale) {
    scale_ = scale;
}

void Scaling::set() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPushMatrix(); // Speichere die aktuelle Matrix
    glScaled(scale_[0], scale_[1], scale_[2]); // Skaliere
}

void Scaling::unset() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPopMatrix(); // Restauriere die vorherige Matrix
}

//
// Rotation
//

Rotation::Rotation(double angle, const Vector& axis)
    : axis_(axis) {
    setAngle(angle);
}

double Rotation::getAngle() const {
    return angle_;
}

const Vector& Rotation::getAxis() const {
    return axis_;
}

void Rotation::setAngle(double angle) {
    angle_ = angle;
    while(angle_ >= 360) { angle_ -= 360; }
    while(angle_ <   0) { angle_ += 360; }
}

void Rotation::setAxis(const Vector& axis) {
    axis_ = axis;
}
```

```
void Rotation::set() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPushMatrix(); // Speichere die aktuelle Matrix
    glRotated(angle_, axis_[0], axis_[1], axis_[2]); // Rotiere
}

void Rotation::unset() {
    glMatrixMode(GL_MODELVIEW); // Aktiviere die Modelview-Matrix
    glPopMatrix(); // Restauriere die vorherige Matrix
}
```

*Attribute.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef ATTRIBUTE_H
#define ATTRIBUTE_H

#include "vector.h"

//
// Attribute
//

class Attribute {
public:
    // set() wird aufgerufen, wenn das Attribute gesetzt wird.
    // Man muss dafuer sorgen, dass man den aktuellen
    // Zustand zwischenspeichert, damit man diesen wieder her-
    // stellen kann, wenn mit unset() dieses Attribute wieder
    // "zurueckgenommen" wird.
    virtual void set() = 0;
    virtual void unset() = 0;
};

//
// Color
//

class Color : public Attribute {
public:
    Color(double red = 0, double green = 0, double blue = 0);
    // Setzt die Farbe auf (red, green, blue)

    virtual void set();
    virtual void unset();

private:
    double color_[3];
};

//
// Style
//

class Style : public Attribute {
public:
    enum FaceStyle { Filled, Outlined, Points };
    Style(FaceStyle frontFace = Filled, FaceStyle backFace = Filled);
    // Setzt den Stiel, mit dem die Polygone gezeichnet werden:
    // Filled:   das Polygon wird ausgefuellt dargestellt
    // Outlined: das Polygon wird umrandet dargestellt
    // Points:   es werden nur die Polygon-Eckpunkte dargestellt
    // Man kann dieses fuer die Vorder- und Rueckseiten getrennt
    // einstellen (siehe hierzu auch GL_CW und GL_CCW).

    virtual void set();
    virtual void unset();

private:
    FaceStyle frontFace_;
    FaceStyle backFace_;
};

#endif // ATTRIBUTE_H
```

*Attribute.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "attribute.h"

#include <iostream.h>
#include <GL/glut.h>

//
// Hilfsfunktionen
//

static inline double clamp(double min, double max, double value) {
    return (value < min) ? min : ((value > max) ? max : value);
}

//
// Color
//

Color::Color(double red, double green, double blue) {
    color_[0] = clamp(0.0, 1.0, red);
    color_[1] = clamp(0.0, 1.0, green);
    color_[2] = clamp(0.0, 1.0, blue);
}

void Color::set() {
    glPushAttrib(GL_CURRENT_BIT); // Merke u.a. die aktuelle Farbe
    glColor3dv(color_); // Setze neue Farbe
}

void Color::unset() {
    glPopAttrib(); // Restauriere die gemerkte Farbe
}

//
// Style
//

Style::Style(FaceStyle frontFace, FaceStyle backFace)
    : frontFace_(frontFace), backFace_(backFace) {
}

void Style::set() {
    // Hinweis: siehe GL_POLYGON_BIT und glPolygonMode()
    cerr << "Style::set() not yet implemented!" << endl;
}

void Style::unset() {
    cerr << "Style::unset() not yet implemented!" << endl;
}
```

*CGApplication.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef CGAPPLICATION_H
#define CGAPPLICATION_H

#include <GL/glut.h>

// einige haeufig verwendete Header-Dateien
#include <assert.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>

class CGApplication {
public:

    // Die Klasse "CGApplication" stellt ein Rahmenprogramm bereit,
    // mit dem sie ihre Aufgaben loesen koennen. Dazu muessen sie
    // eine Klasse hiervon ableiten, und eine oder mehrere der
    // weiter unten deklarierten on*()-Methoden ueberschreiben.
    // Diese Klasse ist ein sogenanntes "Singleton", d.h.: Es kann
    // immer nur eine einzige Instanz von dieser (oder einer
    // abgeleiteten) Klasse erzeugt werden!
    CGApplication();
    virtual ~CGApplication();

    // Definiere die moeglichen Buffer, die von OpenGL angefordert
    // werden koennen:
    enum Buffer { ColorBuffer = 1, DepthBuffer = 2, DoubleBuffer = 4, StencilBuffer = 8 };

    // Startet die Anwendung und muss im Hauptprogramm aufgerufen werden.
    // buffers: fordere diese Buffer von OpenGL an
    // msecsTimer: In diesem Intervall (in Millisekunden) sollen
    //             Time-Events erzeugt werden, fuer die dann die
    //             Methode onTimer() aufgerufen wird. Ein Wert von
    //             0 erzeugt keine Time-Events!
    void start(int argc, char* argv[],
               const char* windowTitle = "CGApplication",
               unsigned int buffers = ColorBuffer | DepthBuffer | DoubleBuffer,
               unsigned int msecsTimer = 0,
               unsigned long windowWidth = 400,
               unsigned long windowHeight = 400);

    // Diese Methode wird nur einmal beim Start der Anwendung aufgerufen
    // und dient zur Initialisierung von OpenGL (hier kann z.B. die
    // Hintergrundfarbe fuer das Anwendungsfenster definiert werden). Sie
    // kann von einer abgeleiteten Klasse ueberschrieben werden.
    virtual void onInit();

    // Jedesmal wenn der Fensterinhalt neu gezeichnet werden muss,
    // wird diese Methode aufgerufen. Sie muss von einer abgeleiteten
    // Klasse ueberschrieben werden!
    virtual void onDraw() = 0;

    // Jedesmal wenn sich die Fenstergroesse geaendert hat, wird diese
    // Methode aufgerufen (die Methode onDraw() wird im Anschluss
    // automatisch aufgerufen). Sie muss von einer abgeleiteten Klasse
    // ueberschrieben werden!
    virtual void onSize(unsigned int newWidth, unsigned int newHeight) = 0;

    // Spezifiziert die beiden Konstanten "LeftMouseButton" und
    // "RightMouseButton".
    enum MouseButton { LeftMouseButton, RightMouseButton };

    // Spezifiziert die beiden Konstanten "MouseButtonDown" und
    // "MouseButtonUp".
    enum MouseButtonEvent { MouseButtonDown, MouseButtonUp };

    // Jedesmal wenn eine Maustaste im Fenster gedrueckt wird, wird diese
```

```
    // Methode mit der entsprechenden Taste ("LeftMouseButton" oder
    // "RightMouseButton") und den zugehoerigen Koordinaten (x und y)
    // aufgerufen. Die Koordinaten werden als Pixel uebergeben, wobei der
    // Koordinaten-Ursprung die linke untere Fensterecke ist. Diese
    // Methode kann von einer abgeleiteten Klasse ueberschrieben werden.
    virtual void onButton(MouseButton button, MouseButtonEvent event,
                          int x, int y);

    // Jedesmal wenn die Maus mit gedrueckter Maustaste ueber das Fenster
    // bewegt wird, wird diese Methode mit den entsprechenden Koordinaten
    // aufgerufen (siehe auch Methode onButton()). Diese Methode kann von
    // einer abgeleiteten Klasse ueberschrieben werden.
    virtual void onMove(MouseButton button, int x, int y);

    // Jedesmal wenn eine Taste gedrueckt wird, wird diese Methode
    // aufgerufen.
    virtual void onKey(unsigned char key);

    // Werden Time-Events angefordert (msecsTimer != 0 in der Methode
    // start()), so wird bei jedem Time-Event diese Methode aufgerufen:
    virtual void onTimer();

    // Diese Methode erzeugt ein Redraw-Event (so dass die Methode
    // onDraw() aufgerufen wird). Man sollte diese Methode verwenden
    // (und nicht direkt onDraw()), um zu signalisieren, dass das Bild
    // neu gezeichnet werden soll!
    void needsRedraw();

    // Diese Methode sollte nach Beendigung aller Zeichenoperationen fuer
    // ein Bild (normaler Weise am Ende der Methode onDraw()) aufgerufen
    // werden, um den Front- mit dem Back-Buffer zu tauschen! Wird dieser
    // Aufruf ausgelassen, so werden die Zeichenoperationen nicht sichtbar!!!
    void swapBuffers();

private:
    // Verbiete den Copy-Konstruktor und den Zuweisungsoperator, da
    // diese Klasse ein Singleton ist und somit nicht kopiert werden
    // darf!
    CGApplication(const CGApplication&);
    CGApplication& operator=(const CGApplication&);
};

#endif // CGAPPLICATION_H
```

*CGApplication.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "cgapplication.h"

// Diese Variable enthaelt einen Zeiger auf die aktive Anwendung.
static CGApplication* activeCGApplication_ = 0;
static CGApplication::MouseButton activeMouseButton_ = CGApplication::LeftMouseButton;

CGApplication::CGApplication() {
    assert(!activeCGApplication_); // Es ist nur eine aktive Anwendung erlaubt!!!
    activeCGApplication_ = this;
}

CGApplication::~CGApplication() {
    activeCGApplication_ = 0;
}

// Auf diese Ereignisse wird default-maessig nicht reagiert:
void CGApplication::onInit() { }
void CGApplication::onButton(MouseButton button, MouseButtonEvent event, int x, int y) { }
void CGApplication::onMove(MouseButton buttoin, int x, int y) { }
void CGApplication::onKey(unsigned char key) { }
void CGApplication::onTimer() { }

// glut-Callback fuer den Zeichenaufruf
static void displayFunc() {
    assert(activeCGApplication_);
    activeCGApplication_->onDraw();
}

// glut-Callback fuer die Aenderung der Fenstergroesse
static void reshapeFunc(int newWidth, int newHeight) {
    assert(activeCGApplication_);
    activeCGApplication_->onSize(newWidth, newHeight);
}

// glut-Callback fuer das Druecken einer Maustaste
static void mouseFunc(int button, int state, int x, int y) {
    assert(activeCGApplication_);
    y = glutGet(GLenum(GLUT_WINDOW_HEIGHT)) - y; // Koordinaten-Ursprung: links unten!!!
    activeMouseButton_ = ((button == GLUT_LEFT_BUTTON)
                          ? CGApplication::LeftMouseButton
                          : CGApplication::RightMouseButton);
    activeCGApplication_->onButton(activeMouseButton_,
                                   ((state == GLUT_DOWN)
                                    ? CGApplication::MouseButtonDown
                                    : CGApplication::MouseButtonUp),
                                   x, y);
}

// glut-Callback fuer das Ziehen bei gedrueckter Maustaste
static void motionFunc(int x, int y) {
    assert(activeCGApplication_);
    y = glutGet(GLenum(GLUT_WINDOW_HEIGHT)) - y; // Koordinaten-Ursprung: links unten!!!
    activeCGApplication_->onMove(activeMouseButton_, x, y);
}

// glut-Callback fuer einen Tastendruck
static void keyboardFunc(unsigned char key, int x, int y) {
    assert(activeCGApplication_);
    activeCGApplication_->onKey(key);
}

// glut-Callback fuer ein Timer-Event
static void timerFunc(int msecs) {
    assert(activeCGApplication_);
    activeCGApplication_->onTimer();
    glutTimerFunc(msecs, timerFunc, msecs);
}
```

```
void CGApplication::start(int argc, char* argv[],
                          const char* windowTitle,
                          unsigned int buffers, unsigned int msecsTimer,
                          unsigned long width, unsigned long height) {
    assert(buffers & ColorBuffer);

    // Initialisiert Glut
    glutInit(&argc, argv);

    // Fordert ein OpenGL-Fenster im RGB-Format mit oder ohne Double-Buffering an:
    glutInitDisplayMode(GLUT_RGB
                        | ((buffers & DepthBuffer) ? GLUT_DEPTH : 0)
                        | ((buffers & DoubleBuffer) ? GLUT_DOUBLE : GLUT_SINGLE)
                        | ((buffers & StencilBuffer) ? GLUT_STENCIL : 0));

    // Legt die Fenstergroesse fest:
    glutInitWindowSize(width, height);

    // Erzeugt das OpenGL-Fenster mit dem entsprechenden Namen:
    glutCreateWindow(windowTitle);

    // Registriere die oben definierten Callbacks fuer die
    // entsprechenden glut-Ereignisse:
    glutDisplayFunc(displayFunc);
    glutReshapeFunc(reshapeFunc);
    glutMouseFunc(mouseFunc);
    glutMotionFunc(motionFunc);
    glutKeyboardFunc(keyboardFunc);

    if(msecsTimer > 0) { glutTimerFunc(msecsTimer, timerFunc, msecsTimer); }

    // Zeige das OpenGL-Fenster auf dem Bildschirm an
    glutShowWindow();

    // Rufe die Methode zur Initialisierung von OpenGL auf:
    onInit();

    // Starte die Verarbeitung der glut-Ereignisse:
    glutMainLoop();
}

void CGApplication::needsRedraw() {
    glutPostRedisplay();
}

void CGApplication::swapBuffers() {
    glutSwapBuffers();
}
```

*CGRobot.h*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#ifndef CGROBOT_H
#define CGROBOT_H

#include "cgapplication.h"
#include "node.h"
#include "transformation.h"

class CGRobot : public CGApplication {
public:
    CGRobot();

    virtual void onInit();
    virtual void onDraw();
    virtual void onSize(unsigned int newWidth, unsigned int newHeight);

    virtual void onKey(unsigned char key);
    virtual void onTimer();

private:
    Node* rootNode_;

    bool pause_;
    bool constructed_;
    bool cameraBezier_;

    Camera*   camera_;
    Rotation* fingerRotation_;
    Rotation* armRotation_;
    Color*    sensorColor_;
};

#endif // CGROBOT_H
```

*CGRobot.cpp*

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 8

#include "cgrobot.h"

CGRobot::CGRobot()
    : pause_(false), constructed_(false), cameraBezier_(true), rootNode_(0) {

    // der komplette Szenengraph enthält nur exakt EINE Box
    Box* box = new Box(Vector(0,0,0), Vector(1,1,1));
    LeafNode* leafBox = new LeafNode(box);

    // ein Finger modellieren
    fingerRotation_ = new Rotation(-1, Vector(0,0,1));
    GroupNode* finger = GroupNode::shareNode(leafBox, 8, new Translation(Vector(1,0,0)),
                                fingerRotation_);

    // Hand erstellen
    GroupNode* hand = GroupNode::shareNode(finger, 5, new Translation(Vector(0,0,1.2)));
    hand->setAttribute(0, new Translation(Vector(1,0.1,3)));
    hand->setAttribute(1, new Rotation(-90, Vector(0,1,0)));
    hand->setAttribute(2, new Scaling(Vector(1/5.8, 1/5.8, 1/5.8)));
    hand->setAttribute(3, new Color(0,0,1));

    // Arm
    GroupNode* oberArm = new GroupNode;
    oberArm->setAttribute(0, new Scaling(Vector(1,0.3,3)));
    oberArm->setChildNode(0, leafBox);

    armRotation_ = new Rotation(45, Vector(1,0,0));
    GroupNode* arm = new GroupNode;
    arm->setAttribute(0, armRotation_);
    arm->setAttribute(1, new Translation(Vector(0,1,0)));
    arm->setAttribute(2, new Scaling(Vector(0.6, 0.6, 0.6)));
    arm->setChildNode(0, oberArm);
    arm->setChildNode(1, hand);

    // beide Arme
    GroupNode* arme = GroupNode::shareNode(arm, 2, new Translation(Vector(1.6,0,0)));
    arme->setAttribute(0, new Translation(Vector(-1.1,0,-1)));
    arme->setAttribute(1, new Color(0, 1, 1));

    // Sensoren
    GroupNode* sensor = new GroupNode;
    sensor->setAttribute(0, new Scaling(Vector(0.2, 0.2, 0.1)));
    sensor->setChildNode(0, leafBox);

    sensorColor_ = new Color(1,0,0);

    GroupNode* sensoren = GroupNode::shareNode(sensor, 2, new Translation(Vector(0.75,0,0)));
    sensoren->setAttribute(0, new Translation(Vector(0,1,0.5)));
    sensoren->setAttribute(1, sensorColor_);

    // Gesicht
    GroupNode* auge = new GroupNode;
    auge->setAttribute(0, new Scaling(Vector(0.2, 0.2, 0.1)));
    auge->setChildNode(0, leafBox);
    GroupNode* augen = GroupNode::shareNode(auge, 2, new Translation(Vector(0.5,0,0)));
    augen->setAttribute(0, new Translation(Vector(0.1,0.7,1)));
    augen->setAttribute(1, new Color(0,0,0));

    GroupNode* mund = new GroupNode;
    mund->setAttribute(0, new Translation(Vector(0.1,0.2,1)));
    mund->setAttribute(1, new Scaling(Vector(0.8, 0.1, 0.1)));
    mund->setAttribute(2, new Color(1,0,0));
    mund->setChildNode(0, leafBox);

    GroupNode* nase = new GroupNode;
    nase->setAttribute(0, new Translation(Vector(0.4,0.5,1)));
    nase->setAttribute(1, new Scaling(Vector(0.2, 0.2, 0.2)));
```

```
    nase->setAttribute(2, new Color(1,1,0));
    nase->setChildNode(0, leafBox);


    // Kopf
    GroupNode* kopf = new GroupNode;
    kopf->setChildNode(0, leafBox);
    kopf->setChildNode(1, sensoren);
    kopf->setChildNode(2, augen);
    kopf->setChildNode(3, mund);
    kopf->setChildNode(4, nase);
    kopf->setAttribute(0, new Translation(Vector(-0.35,1,-0.35)));
    kopf->setAttribute(1, new Scaling(Vector(0.7,0.7,0.7)));
    kopf->setAttribute(2, new Color(1,1,1));

    // Standfläche
    GroupNode* boden = new GroupNode;
    boden->setChildNode(0, leafBox);
    boden->setAttribute(0, new Translation(Vector(-1,-3,-1)));
    boden->setAttribute(1, new Scaling(Vector(2, 0.1, 2)));
    boden->setAttribute(2, new Color(0,1,1));

    // Fuss
    GroupNode* fuss = new GroupNode;
    fuss->setChildNode(0, leafBox);
    fuss->setAttribute(0, new Translation(Vector(-0.35,-3,-0.35)));
    fuss->setAttribute(1, new Scaling(Vector(0.7, 2, 0.7)));
    fuss->setAttribute(2, new Color(0,0,1));

    // Rumpf
    GroupNode* rumpf = new GroupNode;
    rumpf->setChildNode(0, leafBox);
    rumpf->setAttribute(0, new Scaling(Vector(1,2,1)));
    rumpf->setAttribute(1, new Translation(Vector(-0.5,-0.5,-0.5)));
    rumpf->setAttribute(2, new Color(0.5,0.5,0.5));

    // Roboter zusammensetzen
    camera_ = new Camera(Vector(-2,-2,3));
    GroupNode* robot = new GroupNode;
    robot->setAttribute(0, camera_);
    robot->setChildNode(0, arme);
    robot->setChildNode(1, kopf);
    robot->setChildNode(2, fuss);
    robot->setChildNode(3, boden);
    robot->setChildNode(4, rumpf);

    rootNode_ = robot;

    // Szene fertig gebaut, onTimer darf ab jetzt eingreifen
    constructed_ = true;
}



void CGRobot::onInit() {
    glClearColor(0, 0, 0, 1);
    glEnable(GL_DEPTH_TEST);

    // Setze die Beleuchtungseinstellungen:
    static const float specular[]  = { 1.0, 1.0, 1.0, 1.0 };
    static const float shininess[] = { 50.0 };
    glMaterialfv(GL_FRONT, GL_SPECULAR,  specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, shininess);

    static const float lightPosition[] = { 1.5, 1.5, -3.5, 0.0 };
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

    glShadeModel(GL_SMOOTH);
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_NORMALIZE);
}
```

```
void CGRobot::onDraw() {
    // Loesche den Farb-Speicher und den Speicher
    // fuer den Tiefen-Test!
    glClearColor(0.8, 0.8, 0.8, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /**** überschrieben durch Camera !!! ****/

    // Wir sind 3 Einheiten vom Nullpunkt entfernt (auf
    // der z-Achse) und schauen direkt in den Nullpunkt.
    // Die y-Achse soll nach oben zeigen:
/*    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0, 0, 3,  // Position
              0, 0, 0,  // auf diesen Punkt wird geschaut
              0, 1, 0); // Up-Vektor zeigt nach oben
*/

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.5, 1.5, -1.5, 1.5, 1, 6);

    // Zeichne den Szene-Graphen
    if(rootNode_) { rootNode_->traverse(); }

    // Nicht vergessen!!!
    swapBuffers();
}

void CGRobot::onSize(unsigned int newWidth, unsigned int newHeight) {
    glViewport(0, 0, newWidth - 1, newHeight - 1);
}

void CGRobot::onKey(unsigned char key) {
    const unsigned char ESC   = 27;
    const unsigned char SPACE = ' ';

    switch (key)
    {
    case SPACE: pause_ = !pause_;
                break;
    case 'c':
    case 'C':   cameraBezier_ = !cameraBezier_;
                break;

    case ESC:   exit(0);
    }
}

void CGRobot::onTimer() {
    if (!constructed_)
        return;

    static int nTimerTicks = 0;

    // durchgängiger Timer
    nTimerTicks++;

    // Bewegung
    if(!pause_)
    {
        // Kamera bewegen
        if (cameraBezier_)
        {
            // auf Bezierkurve
            const int bezierPoints = 4;
            Vector v[bezierPoints] = { Vector(-2,-2, 3),
                                       Vector( 5, 2, 7),
                                       Vector( 3, 5,-5),
                                       Vector(-2,-2,-2) };

            static double t = 0.01;
            static double incT = 0.01;
```

```
            if (t<0.01 || t>0.99)
                incT = -incT;
            t += incT;
            camera_->setFrom(bezier(v,bezierPoints,t));
        }
        else
        {
            // auf vordefinierter Bahn
            double angle = nTimerTicks*3.1415926/180.0;
            camera_->setFrom(Vector(-3*sin(angle), 3*sin(angle), 3*cos(angle)));
        }

        // Finger öffnen/schliessen
        static double fingerAngleInc = -1;
        double angle = fingerRotation_->getAngle();
        if (angle == 340 || angle == 0)
            fingerAngleInc = -fingerAngleInc;
        fingerRotation_->setAngle(angle+fingerAngleInc);

        // Arme bewegen
        static double armAngleInc = -1;
        angle = armRotation_->getAngle();
        if (angle == 80 || angle == 30)
            armAngleInc = -armAngleInc;
        armRotation_->setAngle(angle+armAngleInc);
    }

    // blinken
    delete sensorColor_;
    if (nTimerTicks % 2 == 0)
        sensorColor_ = new Color(1,0,0);
    else
        sensorColor_ = new Color(1,0.8,0.8);

    needsRedraw();
}

int main(int argc, char* argv[]) {
    cout << "SPACE toggles rotation" << endl
         << "ESC quits the program" << endl;

    CGRobot CGRobot;
    CGRobot.start(argc, argv, "CGRobot, Stephan Brumme, 702544",
                  CGApplication::ColorBuffer
                | CGApplication::DepthBuffer
                | CGApplication::DoubleBuffer,
                25);
    return 0;
}
```