Übung 1 - Computergrafik-I, WS 2015/16

Christoph Stumpe, Fabian Wendland, Martin Zier
Beuth Hochschule für Technik Berlin

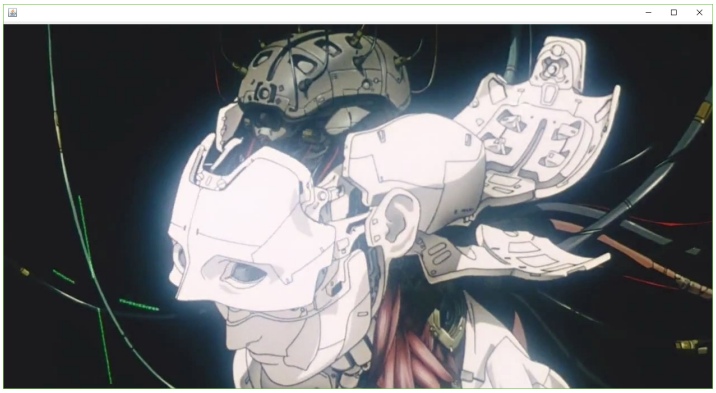

Abbildung 1: *Vorbereitungsübung 1 – Quelle:* GHOST IN THE SHELL*, 1996*

Inhaltsverzeichnis

1 Einführung

In der Vorbereitung werden Grundkenntnisse überprüft und dienen zur Wiederholung der objektorientierten Programmierung. Außerdem wird auf dem vorbereiteten Quellcode aufgebaut und programmiert.

2 Aufgabenstellung

Alle dargestellten Aufgaben beziehen sich auf die Implementierung mit Java sofern nicht anders gekennzeichnet.

2.1 Bilddarstellung & -generierung

Es werden zwei verschiedene Programme gefordert:

- **Bilddarstellung**: Ein Programm, welches nach dem Pfad einer Bilddatei fragt und diese dann in einem Fenster anzeigt.
- **Bildgenerierung**: Ein Programm, welches in einem Fenster mit schwarzem Grund eine rote Linie im 45°Winkel zeigt. Der gezeigte Inhalt entspricht der Fenstergröße und muss gegebenenfalls nachgezeichnet werden, wenn die Fenstergröße verändert wird. Dieses generierte Bild soll in seiner aktuellen Rendergröße zu speichern sein — das bedeutet ebenfalls, dass bei veränderter Größe das gespeicherte Bild sich in Größe und Proportion verändert.

2.2 Grundlegende Vektormathematik

Die mathematischen Implementierungen werden wie folgt gefordert[1]:

- **Normal3**: Eine Normale mit x, y, z Koordinate.
  Ebenfalls soll sie mit einer Normale multipliziert und addiert werden können.
  Außerdem soll das Skalarprodukt mit einem *Vector3* möglich sein.

[1] Jedes Element der Auflistung bezieht sich auf ein einzelnes Objekt in eigener Klasse.

- **Point3**: Ein Punkt mit x, y, z Koordinate.
  Es soll möglich sein einen *Point3* mit einem *Vector3* zu addieren und zu subtrahieren. Das Ergebnis ist ein neuer *Point3*.
  Außerdem ergibt die Subtraktion mit einem *Point3* einen neuen *Vector3*.
- **Mat3x3**: Eine 3x3 Matrix, die über einen vollständigen Konstruktor initialisiert wird. Zusätzlich soll die Determinante bei Initialisierung bestimmt werden, falls nicht mit dem Konstruktor aufgerufen.
  Diese Matrix soll Methoden haben um einzelne Zeilen (mittels eines *Vector3*) auszutauschen.
  Zusätzlich sollen folgende Matrixmultiplikationen möglich sein [2]:
  1. **Mat3x3** :: *Mat3x3* » *Mat3x3*
  2. **Mat3x3** :: *Vector3* » *Vector3*
  3. **Mat3x3** :: *Point3* » *Vector3*

2.3 Pythonimplementierung

Wir haben uns entschieden den Basis-Raytracer in Python zu portieren. Bis zur Erstellung dieses Papiers ist noch nicht endgültig entschieden ob wir das Render-Backend in C (mit Python-Bindings) programmieren oder gar in Assembler schreiben. Allein der Geschwindigkeit wegen, wird es sich im späteren Verlauf als sinnvoll darstellen, dass alle einfachen mathematischen Objekte in C implementiert werden und die Python-Portierung nur noch die dazugehörigen API offengelegt wird.
Die bisherige Implementation wird den advanced Python-Syntax verwenden, indem wir z. B. die Standardoperatoren von Python-Objekten überschreiben.

Die Debug-Output aller Objekte und der notwendigen Funktionen wird gespiegelt zur Java-Implementierung programmiert. Dabei werden ggf. Segmente so portiert, dass syntaktisch und funktional alle Richtlinen der PEP8[3] eingehalten werden.

2.4 Addendum

Die Klassen werden durch Akzeptanzkriterien der vorliegenden Aufgaben geprüft. Diese Akzeptanzkriterien können im Aufgabenblatt zu diesem Papier nachgesehen werden. Eine Kopie der Aufgaben ist auf der Seite von [4] erhältlich. Unittests werden nicht geschrieben.

Der dazugehörige Python-Code wird durch die selbigen Akzeptanzkriterien geprüft.

3 Lösungsstrategien

Die Vorbereitungsaufgaben benötigten keine gesonderten Strategien. Unsere Übungsgruppe hat sich darauf geeinigt, zunächst einen eigenen Branch anzulegen und die Vorbereitungsaufgaben selbst zu implementieren. Dadurch können nach der Semesterpause noch einmal die Kenntnisse aufgefrischt werden und die Aufgaben stellen ein Mindestmaß des bisherigen Grundstudiums dar.

[2] Matrix :: Eingabe » Ausgabe

[3] `https://www.python.org/dev/peps/pep-0008/`

[4] `http://rehfeld.beuth-hochschule.de/2015-ws-cg1.html`

4 Implementierung und Bearbeitungszeit

Die Java-Implementierung versucht alle Code-Conventions einzuhalten und entspricht der Aktualität zur Übungsabgabe. Diese Implementierung wird nach bestem Wissen und Gewissen zu dem Zeitpunkt der Abgabe von den Studenten programmiert.

4.1 Bilddarstellung

Bei der Implementierung kann ein einfaches, unmodifiziertes `JOptionPane` verwendet werden. Nach der Initialisierung aus der Auswahl des `JOptionPane` wird eine Gegenprobe auf den Dateityp durchgeführt — mithilfe von `java.nio.file.Files.probeContentType`. Es wird ein *MIME*-String ausgegeben, anhand dessen herausgefunden werden kann, ob es sich um eine Bilddatei handelt. Danach wird das Bild in ein `BufferedImage` geladen und in einem `JFrame` angezeigt.

Bearbeitungszeit: 0,5 – 1,5 Stunden

4.2 Bildgenerierung

Ein `JFrame` erzeugt ein `ImagePanel`. Dieses `ImagePanel` ist ein modifizierted `JPanel` welches eine rote Linie im 45°Winkel erzeugt — die Bildinformationen werden in einem `BufferedImage` gespeichert. Durch die Implementierungsform von `BufferedImage` ist die erzeugte Bildinformation grundsätzlich schwarz. Diese Gegebenheit lässt sich dadurch ausnutzen, dass für die Liniengenerierung nur noch das größere Maß von Höhe und Breite definiert werden muss und erhält dadurch eine Laufzeit von $\mathcal{O}(n)$[5] statt $\mathcal{O}(n_h * n_w)$[6].

Bearbeitungszeit: 1,0 – 1,5 Stunden

4.3 Vektormathematik

Die Implementierung erfolgt nach den Richtlinien aus 2.2. Die Implementierung ist simpel und geradlinig nach den UML-Diagrammen aus der Aufgabenstellung programmiert.

Bearbeitungszeit: 2,0 – 2,5 Stunden

4.4 Pythonimplementierung

Wie vorher bereits erwähnt, erfolgt die Implementierung mit ähnlicher Objekt-API wie der Java-Implementation. Zwar werden Objekt-Operationen überschrieben, allerdings werden auch darauf Operationen auf diese Objekt-Methoden weitergeleitet:

```
class Object:
        def __mul__(self, other):
                pass
        def mul(self, other):
                return self * other
```

Dadurch wird der Portierungsvorgang im späteren Verlauf erheblich vereinfacht, da nicht alle Funktionsaufrufe neu geschrieben werden müssen und unübersichtliche Rechnungen können erheblich verkürzt werden.

Bearbeitungszeit: 2 Stunden

[5] `max(height, width)`

[6] `height * width`

5 Aufgetretene Probleme der Implementierung

- **LaTeX- Einarbeitung, Einrichtung**:
  LaTeX stellt eine ungeheure Herausforderung dar, hauptsächlich da keine empfohlenden Tools vorgestellt wurden. Die Arbeit ist allen beteiligten Studenten neu.

6 Quellen & Literaturverzeichnis

- `https://asalga.wordpress.com/2012/09/23/understanding-vector-reflection-visually/`
- `http://geomalgorithms.com/points_and_vectors.html#Vector-Addition`
- `http://docs.oracle.com/javase/8/docs/api/`
- Fundamentals of Computer Graphics – PETER SHIRLEY & STEVE MARSCHNER