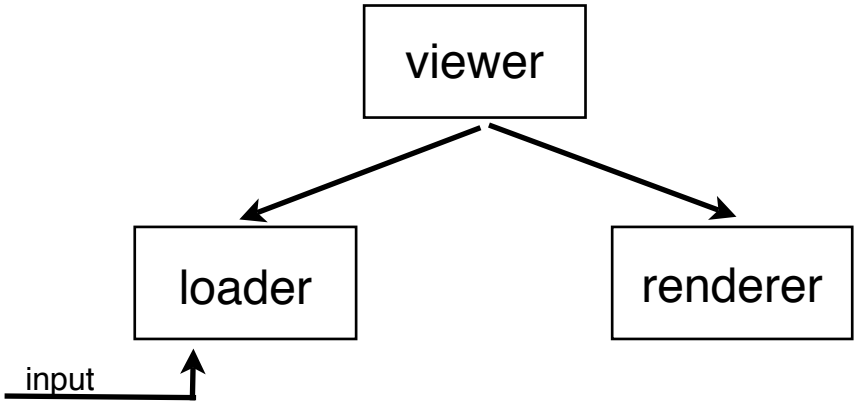
Vorlesung Softwaretest und -debugging

Dr. Carsten Gnörlich cg@techfak.uni-bielefeld.de

Übungsblatt 6 8. Jun. 2012

Beispielprogramm zum Integrationtest

Für die Grafikbibliothek wurde ein Anzeigeprogramm für komplexe 3D-Modelle entwickelt, das den folgenden Modulgraphen besitzt:



Das viewer-Modul entspricht im Wesentlichen dem Grundgerüst zum Arbeiten mit der Grafikbibliothek aus den vorherigen Übungen.

Das loader-Modul exportiert einen Datentyp *Model*, der die Beschreibung eines 3D-Modells enthält. Mit Hilfe der Funktion *LoadModel()* wird eine Datei mit einer Modellbeschreibung eingelesen und in Form eines *Model*-Objektes zurückgegeben.

Das renderer-Modul nimmt ein *Model*-Objekt entgegen und zeichnet dieses in einen ebenfalls anzugebenen *FrameBuffer* aus der Perspektive einer angegebenen *View.* Die entsprechende Funktion heißt *RenderModel().*

Die zugehörigen Quellkode-Dateien sind *viewer.c, viewer-load.c* und *viewer-render.c*. Eine Beispieldatei für ein Modell ist *beispiel.model.*

Im Übungsverzeichnis liegen die Dateien bereits im integrierten Zustand vor, d.h. man kann sie direkt übersetzen und ausführen.

Aufgabe 1

Die Module seien mit Hilfe eines absteigenden Tests zu integrieren. Ersetzen Sie die Module *loader* und *renderer* durch Platzhalter, die lediglich per *printf()* ausgeben, daß sie aufgerufen wurden. Das *loader*-Modul soll außerdem den Namen der Datei ausgeben, mit dem es aufgerufen wurde.

Eine Beispielausgabe des modifizierten Programms wäre:

...

LoadModel(beispiel.model) aufgerufen.

Render() aufgerufen.

Aufgabe 2

Im Zuge eines Modultests soll das Modul *loader* getrennt getestet werden. Entwerfen Sie für das Modul einen Treiber, der es erlaubt das Modul mit einem Dateinamen aufzurufen. Außerdem soll der Treiber die zurückgegebene Datenstruktur des Modells auszugsweise ausgeben. Dazu soll es zu jedem im Modell enthaltenen Baustein (Struktur vom Typ *Brick*) folgende Eigenschaften ausgeben:

brick->id

brick->pos (der Positionsvektor des Bausteins)

brick->shape->classArgs

brick->material->classArgs

Beispielausgabe:

```
Baustein 3, Position  8.00 -5.00  1.50
  Form    :  Arc 9 12 2 7 19 1 1 bf
  Material:  Chaos  1.11 ffffff ffffff a6a6a6 ffffff ffffff 320 417 651

Baustein 4, Position  8.00 -5.00 10.50
  Form    :  Arc 9 12 2 7 19 1 1 bf
  Material:  Chaos  1.11 ffffff ffffff a6a6a6 ffffff ffffff 320 417 651

Baustein 6, Position  9.00 -12.00  1.50
  Form    :  Slope 4 2 9 1 2 ff
  Material:  Chaos  1.11 ffffff ffffff a6a6a6 ffffff ffffff 320 417 651
...
```

Hinweise:

- Verwenden Sie das Programmgerüst *treiber.c*, da darin schon die notwendigen Initialisierungen und Aufräumarbeiten für die Grafikbibliothek enthalten sind.
- Die Struktur des Datentyps *Model* können Sie in der Datei *viewer-load.h* nachlesen. In */vol/lehre/softwaretest/include/3dlibrary.h* finden Sie Informationen über den Aufbau der *Brick*-Datenstruktur.
- In der Funktion *free_model()* der Datei *treiber.c* finden Sie ein Beispiel zum Durchlaufen der Liste der Bausteine eines Modells.

Aufgabe 3

Im Verlauf des Integrationstests seien die Module *viewer* und renderer integriert worden. Erzeugen Sie mit Hilfe eines Platzhalters ein Modul-Objekt, das genau einen Baustein enthält, und testen Sie damit das Teilsystem.

Hinweise:

Eine minimal lauffähige Initialisierung der Modell-Datenstruktur benötigt die folgenden Werte:

SetQuaternion(model->view, 0.25, 0.25, 0.066987, 0.933013);

model->zoom = 16.0;

Für die Initialisierung der Brick-Datenstruktur wird folgendes benötigt:

IdentityQuaternion(brick->otn);

brick->shape = CreateShape(*shape-spec*);

UpdateFaceInfo(brick);

brick->material = CreateMaterial(*material-spec*);

UpdateMaterialClosure(brick);

Vergessen Sie nicht, den erzeugten Baustein in das Modell einzutragen. Passende Werte für *shape-spec* und *material-spec* können Sie der Datei *beispiel.model* entnehmen bzw. Sie haben diese Werte schon mit der Ausgabe von Aufgabe 2 gesehen und erzeugt.