Malprogramm für

High Dynamic Range Bilder

Studienarbeit

vorgelegt von
Andreas Thun

Institut für Computervisualistik
Arbeitsgruppe Computergrafik

Betreuer: Thorsten Grosch
Prüfer: Prof. Dr. Stefan Müller

März 2004

Inhaltsangabe

1. Einleitung

Gängige Malprogramme verwenden zumeist Formate, bei denen die Farbwerte für jedes Pixel auf ein Byte je Farbkanal beschränkt sind. Damit sind dann für den Rot-, Grün- und Blau-Kanal jeweils nur 256 Abstufungen möglich. Diese Formate werden als Low Dynamic Range Bilder bezeichnet, da der Dynamikbereich, also der Kontrast zwischen der hellsten und der dunkelsten Stelle im Bild, stark begrenzt ist. Die Computergrafik braucht aber für Aufgaben, bei denen beispielsweise fotorealistische Ergebnisse erzielt werden sollen, oft High Dynamic Range Bilder, die für jeden Farbkanal eine Gleitkommazahl zum Speichern der Farbintensität bereitstellen.

In dieser Studienarbeit wird ein Malprogramm, welches High Dynamic Range Bilder bearbeiten kann, implementiert. Im folgenden Kapitel wird das Programm zunächst für einen Anwender in seiner Funktionalität beschrieben. Kapitel 3 betrachtet das Programm aus einer technischen Sicht, indem es sämtliche zugrundeliegende Klassen beschreibt. Im vierten Kapitel werden die Schritte, die nötig sind um das Programm zu starten, im Detail erklärt. Den Abschluss bildet dann der Anhang, in dem die Tastaturkürzel aufgezählt werden, ein Glossar eine Übersicht über die wichtigsten Begriffe bietet, der Inhalt der CD aufgezählt wird und die Referenzen aufgelistet werden.

2. Programmbeschreibung

In diesem Kapitel wird das Malprogramm und seine Funktionalität beschrieben.

2.1 Neues Bild erstellen, vorhandene Bilder öffnen

Um ein Bild zu bearbeiten, muss man zunächst ein neues Bild erstellen oder ein vorhandenes Bild aus einer PFM-Datei laden. Dabei werden grundsätzlich zwei Arten von Bildern unterschieden: einzelne Bilder und Environment Maps. Eine Environment Map besteht aus sechs Einzelbildern, die alle gleich groß und quadratisch sein müssen.

Abbildung 2.1: Datei-Menü

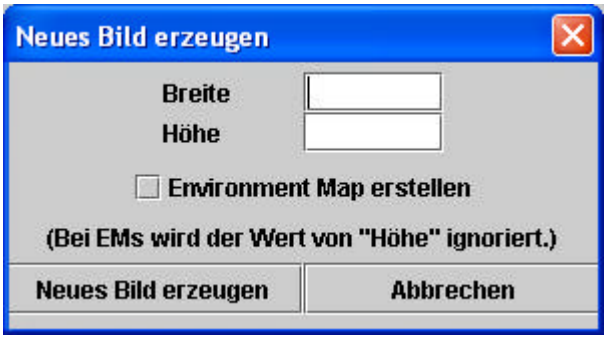


Abbildung 2.2: Dialog zum Erzeugen neuer Bilder

Beide Arten von Bildern können neu über das Menü `Datei > Neues Bild` oder die Tastenkombination `Strg-N` erstellt werden. Daraufhin öffnet sich ein Dialog, bei dem man die Breite und Höhe in Pixeln für das Bild angeben kann. Soll eine Environment Map erstellt werden, muss man das in dem Dialog ankreuzen. In dem Fall kann nur noch eine Größe angeben werden, die dann der Breite und Höhe jedes Quadranten in der Environment Map entspricht. Ein Quadrant entspricht dabei einem der erwähnten Einzelbilder.

Beim Öffnen vorhandener Bilder wird im Menü zwischen Einzelbildern und Environment Maps unterschieden. Ein einzelnes Bild kann man über das Menü `Datei > High Dynamic Range Bild öffnen` oder die Tastenkombination `Strg-H` laden. In dem sich öffnenden Dialog kann man nun eine PFM-Datei im Dateisystem suchen und auswählen.

Bei Environment Maps hingegen muss der Menüpunkt `Datei > Environment Map öffnen` oder die Tastenkombination `Strg-E` genommen werden. Dann öffnet sich ein Dialog in dem man sechs gleich große, quadratische PFM-Dateien angeben muss.

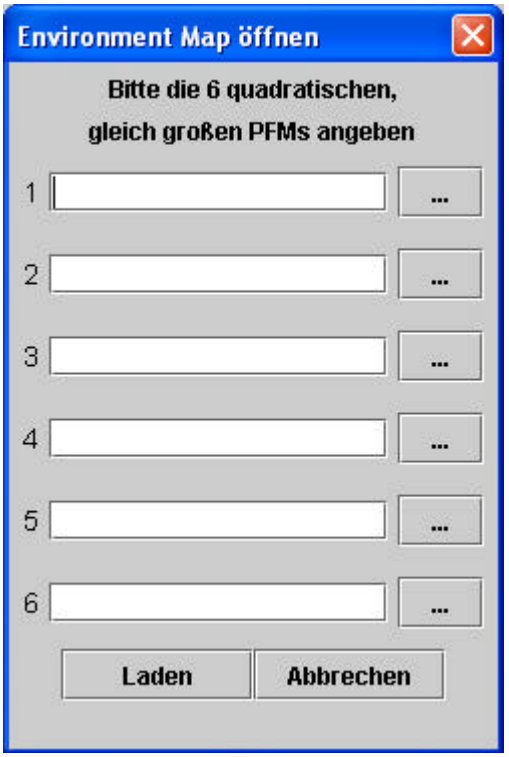


Abbildung 2.3: Dialog zum Öffnen von Environment Maps

2.2 Die Programmoberfläche

Beim Start des Programms steht nur das Dateimenü zu Verfügung. Hier muss, wie schon beschrieben, ein neues Bild angelegt oder ein vorhandenes geöffnet werden. Ist dies geschehen, wird auch die restliche Benutzerschnittstelle angezeigt und die verfügbaren Optionen im Menü ändern sich. Im Dateimenü kann man keine weiteren Bilder öffnen, bis das aktuelle geschlossen ist. Es gibt aber nun die Möglichkeit, das aktuelle Bild abzuspeichern oder zu exportieren. Im Ebenenmenü kann man verschiedene Dreh- und Spiegeloperationen auswählen, die später noch beschrieben werden. Hat man eine Environment Map geöffnet, kann man auch den Quadranten, auf den besagte Operationen ausgeführt werden sollen, auswählen. Am linken Rand der Oberfläche befindet sich der Werkzeugkasten. Hier kann ein Reiter gewählt werden, der dann verschiedene Werkzeuge zur Auswahl und Informationen anzeigt. Der zentrale Teil der Oberfläche dient der Anzeige des Bildes, welches mit einem ein Pixel großen, schwarzen Rand umgeben ist. Durch Benutzung der linken Maustaste kann hier mit dem gerade ausgewählten Werkzeug gearbeitet werden. Im unteren Teil werden in der Informationsleiste die Bildgröße und die Position des Mauszeigers

angezeigt. Hat man eine Environment Map geöffnet, so kann man hier auch den Button zum Umschalten vom bisherigen 2D- in den 3D-Modus nutzen.

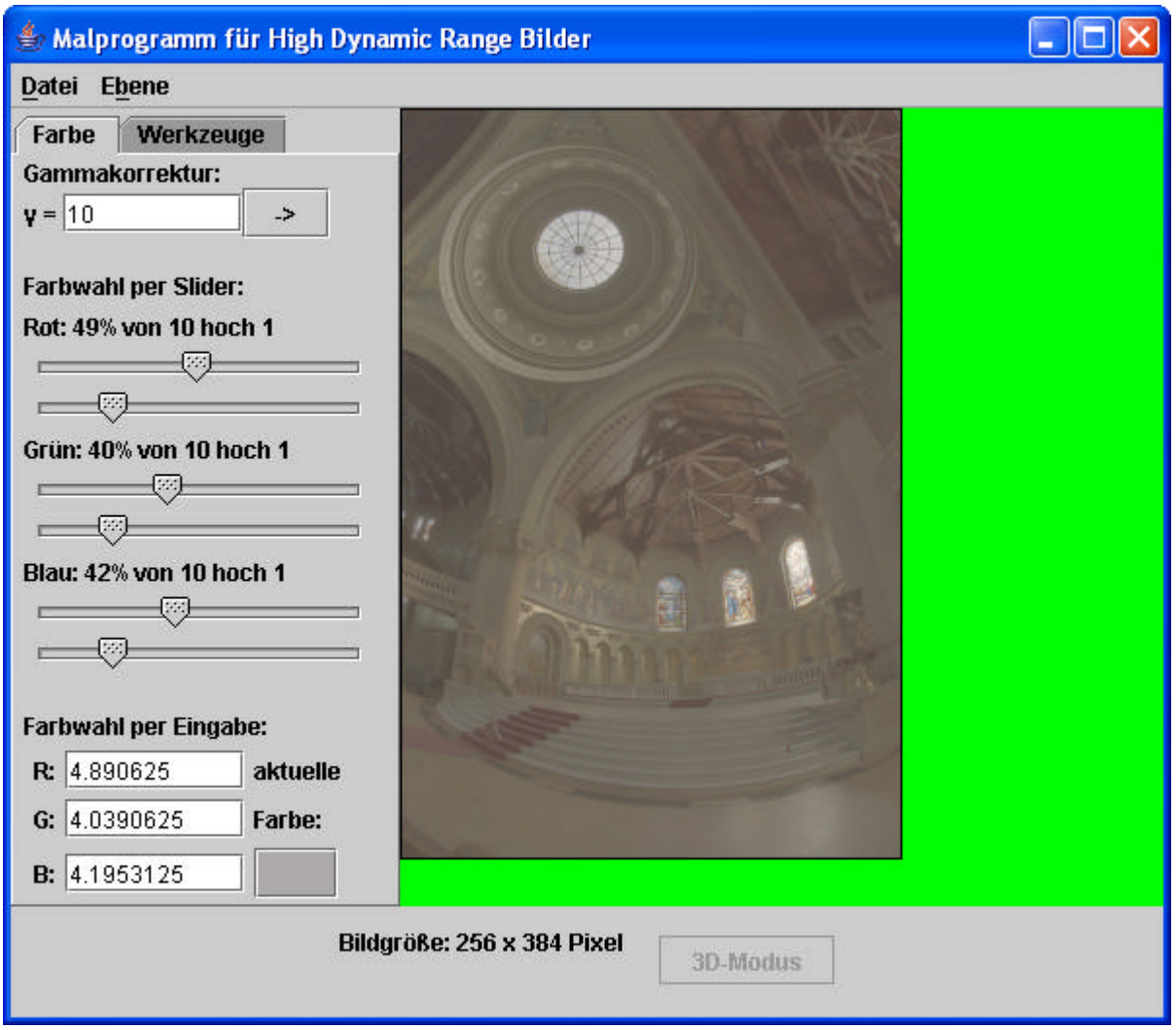


Abbildung 2.4: Screenshot mit geöffnetem High Dynamic Range Bild

Im 3D-Modus wird die Environment Map als Textur auf einen Würfel gelegt. Die untere Informationsleiste hat sich nun auch geändert. Zur besseren Orientierung lassen sich die Koordinatenachsen des Würfels einblenden und mit den Buttons kann man den Würfel um diese Achsen drehen. Die weiteren Buttons dienen dem Heran- und Wegzoomen. Der Reset-Button verwirft alle bisherigen Rotationen und Translationen des Würfels und bringt ihn wieder in die Ausgangsposition. Alternativ zu den Buttons kann man auch die Maus zur Navigation benutzen. Hält man die rechte Maustaste gedrückt, kann man, indem man die Maus nach vorne bewegt, heranzoomen bzw. durch eine umgekehrte Mausbewegung wegzoomen. Hält man die mittlere Maustaste gedrückt, kann man den Würfel rotieren. Diese Rotation ist aber unterschiedlich zu

der mit den Buttons. Mit den Buttons wird der Würfel um die Achsen seines örtlichen Koordinatensystems rotiert. Bei gedrückter mittlerer Maustaste und gleichzeitigen Bewegungen der Maus kann der Würfel um die X- und Y-Achse des Kamerakoordinatensystems rotiert werden. Wie im 2D-Modus dient die linke Maustaste der Anwendung der Werkzeuge. Mit dem Button ganz rechts in der Informationsleiste kann wieder zurück in den 2D-Modus geschaltet werden.

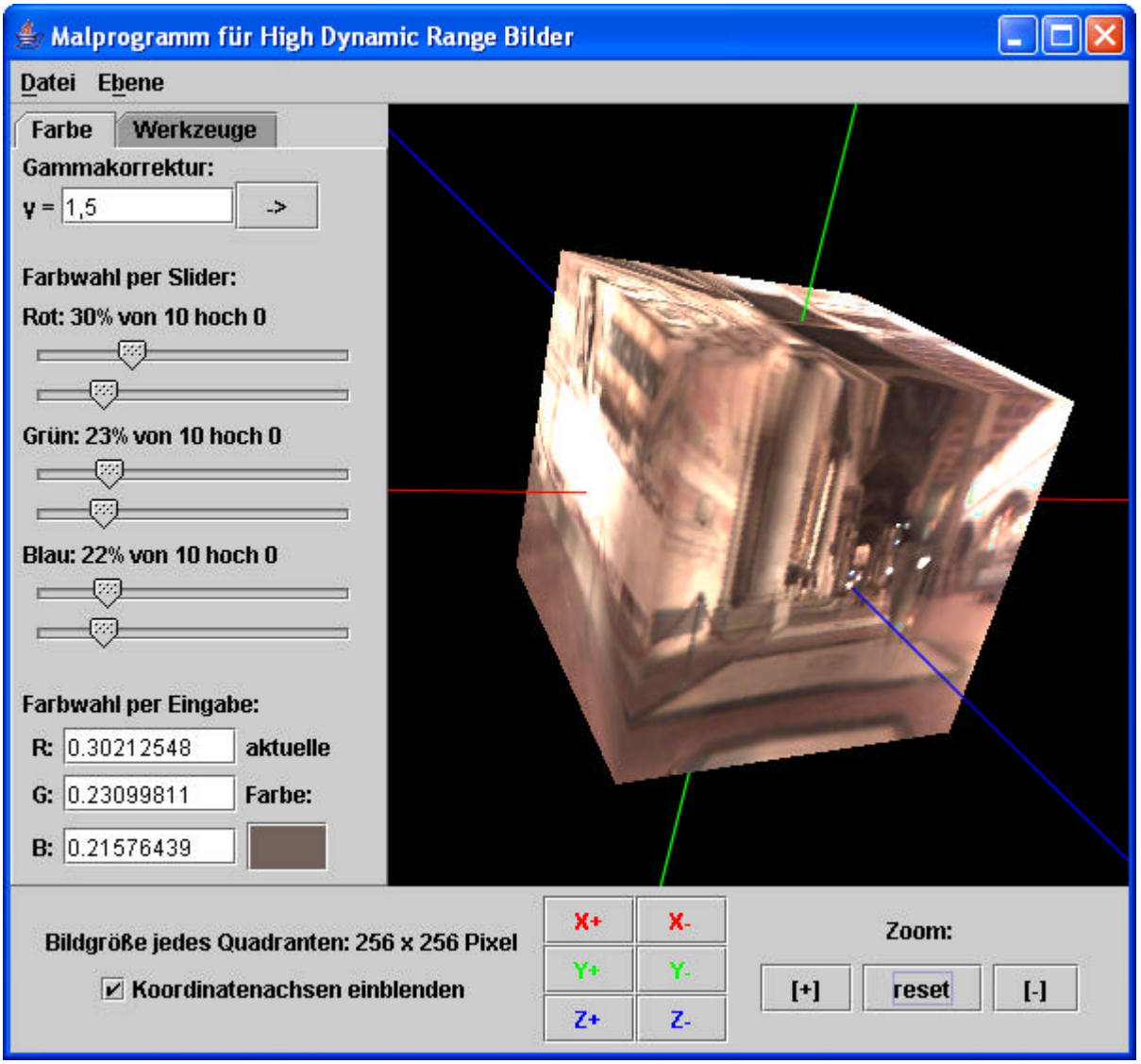


Abbildung 2.5: Screenshot vom 3D-Modus bei geöffneter Environment Map

2.3 Gammakorrektur

Durch die Gammakorrektur wird ein einfaches Tonemappingverfahren zur Darstellung der High Dynamic Range Bilder realisiert. Ziel ist es, das gesamte HDR-Farbspektrum auf Standard RGB-Werte von 0 bis 255 je Kanal zu reduzieren, um das Bild auf dem Monitor darstellen zu können.

Dazu muss der Gammawert wie folgt gewählt werden:

$0 < g < 1$	Das Bild wird abgedunkelt.
$g = 1$	Es findet eine lineare Skalierung statt.
$g > 1$	Das Bild wird aufgehellt.

Bei diesem Tonemappingverfahren wird bei der linearen Skalierung der High Dynamic Range Farbraum in 256 gleich große Bereiche aufgeteilt und jedem dieser Bereiche wird genau ein Low Dynamic Range Farbwert zugeteilt. Bei Werten kleiner als eins ist die Auflösung der Bereiche mit niedrigen Werten größer, so dass das Low Dynamic Range Bild dunkler wird. Analog wird das Bild bei Werten größer als eins aufgehellt, weil die Auflösung hier für Bereiche mit höheren Werten größer ist.

Standardmäßig ist ein Gammawert von vier eingestellt, der sich beim Testen als gute Größe erwiesen hat. Einstellen kann man die Werte, indem man im Werkzeugkasten unter dem Reiter „Farbe“ den Gammawert einträgt und nebenstehenden Button anklickt. Es ist im Übrigen egal, ob man bei der Eingabe einen Punkt oder ein Komma für die Abtrennung der Nachkommastellen benutzt.

2.4 Farbwahl

Die Farbe, mit der gemalt werden soll, kann auf unterschiedliche Weise festgelegt werden. Wählt man den Farbe-Reiter im Werkzeugkasten, so kann man eine beliebige Farbe auf zwei Arten festlegen. Die erste Möglichkeit ist die Nutzung der Slider. Hier stehen für jeden Farbkanal zwei solcher Slider bereit. Der erste gibt einen Prozentwert zwischen 0 und 100 an, der mit der Zehnerpotenz, die mit dem zweiten Slider gewählt werden kann, multipliziert wird. Die Größenordnung des zweiten Sliders kann zwischen 10^{-10} und 10^{38} gewählt werden. Die zweite Möglichkeit ist die Direkteingabe der Farbwerte. In die drei Felder kann man beliebige, nichtnegative Zahlwerte eingeben, die man durch Anklicken des Farbvorschaubuttons auswählt. Dieser Button zeigt immer die Farbe an, mit der gerade gemalt werden kann. Mit der Direkteingabe können natürlich feinere Abstufungen der Farbwerte als mit den Slidern erzielt werden, da diese nur auf ein Hundertstel genau sind. Ändert man einen Farbwert mit den Slidern, so aktualisieren sich die Farbwerte in den Eingabefeldern und umgekehrt. Bei der Direkteingabe ist es egal, ob man einen Punkt oder ein Komma für die Abtrennung der Nachkommastellen benutzt.

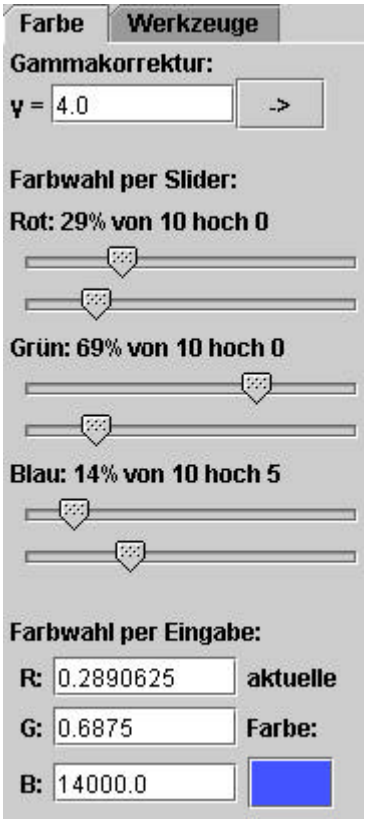


Abbildung 2.6: Reiter mit Gammakorrektur und Farbeinstellungen

Wählt man den Werkzeuge-Reiter im Werkzeugkasten, so kann man die Farbpipette auswählen, die die dritte Möglichkeit der Farbwahl darstellt. Mit dieser ist es möglich, durch Klicken auf das Bild eine bereits im Bild vorhandene Farbe auszuwählen. Die ausgewählte Farbe wird auch durch die Slider und Eingabefelder des Farbe-Reiters angezeigt.

Wenn man nun eine Farbe wählt, die größer als der derzeitige Maximalwert im Bild ist, gibt es zwei Seiteneffekte. Der erste Effekt betrifft den Farbvorschaubutton. Die Farbvorschau stimmt zwar für diesen neuen Maximalwert, aber nicht mehr für den im Bild. Der zweite Effekt ist, dass sobald man mit der neuen Farbe im Bild malt, die

Gammakorrektur nun ein neues Maximum aus dem Bild erhält, so dass es zu einer plötzlichen Abdunkelung des gesamten Bildes kommt.

Löscht man umgekehrt durch beispielsweise Übermalen den bisherigen Maximalwert aus dem Bild, findet keine plötzliche Aufhellung statt. Um die Gammakorrektur nun wieder mit korrektem Maximalwert zu berechnen, muss man einfach den Button für die Gammakorrektur anklicken. Danach sollte die erwartete Aufhellung stattfinden.

2.5 Werkzeuge

Abbildung 2.7: Reiter mit Werkzeugen

Als Werkzeuge sind nur die schon erwähnte Farbpipette sowie ein Stift implementiert. Die Werkzeuge kann man auswählen, indem man den gleichnamigen Reiter aus dem Werkzeugkasten auswählt. Das aktuell ausgewählte Werkzeug ist durch rote Schrift auf dem zugehörigen Button kenntlich gemacht. Für den Stift kann man noch auswählen, wie groß die zu bemalende Fläche sein soll. Wenn man nun auf der Zeichenfläche mit ausgewähltem Stift auf das Bild klickt, wird um die Position des Mauszeigers ein Quadrat mit der Seitenlänge der ausgewählten Größe in Pixeln gemalt, wobei die Position des Mauszeigers den Mittelpunkt des Quadrates darstellt und die Farbe der aktuellen Farbe entspricht, die man unter dem Farbe-Reiter nachsehen kann.

2.6 Drehen und Spiegeln

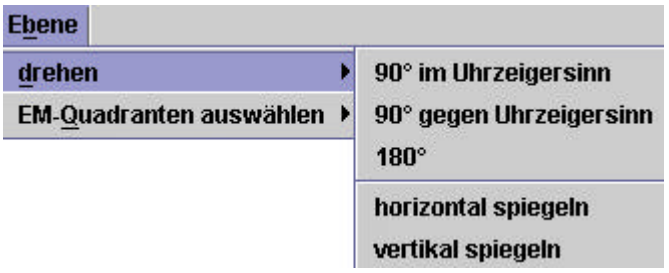
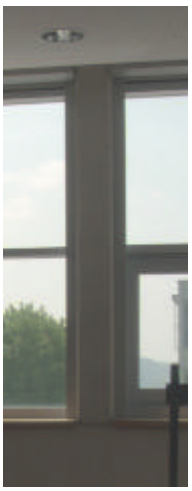


Abbildung 2.8: Menü zum Drehen und Spiegeln

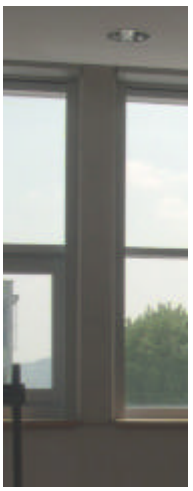
Über den Menüpunkt `Ebene > drehen` kann man jedes Bild auf fünf verschiedene Arten drehen oder spiegeln. Dabei wird wieder zwischen einzelnen Bildern und Environment Maps unterschieden, denn bei letzteren macht es natürlich keinen Sinn, die ausgewählte Operation auf das gesamte Bild anzuwenden.

Im Folgenden ist beispielhaft dargestellt, wie man ein einzelnes Bild drehen und spiegeln kann.

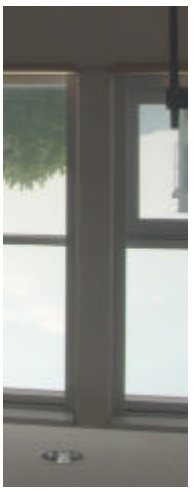
Original



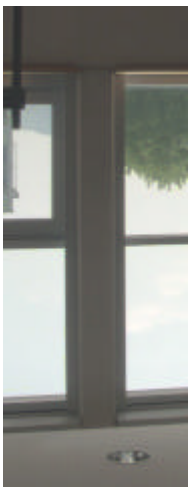
Vertikale Spiegelung



Horizontale Spiegelung



180°-Drehung



90°-Drehung im Uhrzeigersinn

90°-Drehung gegen den Uhrzeigersinn

Abbildungen 2.9: Beispiele zum Drehen und Spiegeln

Die vorgesehenen Ebenen sind nicht mehr ganz implementiert, aber dennoch beim Drehen und Spiegeln berücksichtigt. Bei Drehungen um 90° gibt es bei nichtquadratischen Bildern das Problem, dass eine derart gedrehte Ebene nicht mehr die gleiche Breite und Höhe wie das Bild hat. Daher kann man sich bei solchen Drehungen entscheiden, wie genau die Drehung gemacht werden soll, damit die Ebene wieder ins Bild passt. Entweder wird

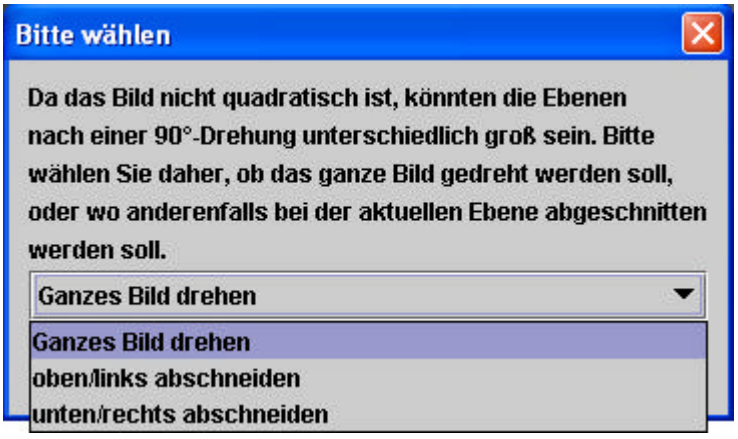


Abbildung 2.10: Dialog mit Detailangaben zu bestimmten Drehungen

das gesamte Bild und damit auch jede Ebene gedreht (so ist es in dem vorausgehenden Beispiel geschehen), oder Bild und Ebenen bleiben in ihrer Größe unverändert und die zu drehende Ebene wird beschnitten. Das Abschneiden hängt nun ferner davon ab, ob die Bildbreite größer als die Bildhöhe ist und es sich somit um ein Querkantbild handelt, oder ob das Bild umgekehrt höher als breit ist und es sich somit um ein Hochkantbild handelt. Im ersteren Fall kann man sich nun entscheiden, ob die Ebene nach der Drehung oben oder unten abgeschnitten werden soll. Beim Hochkantbild dagegen muss man sich zwischen links oder rechts abschneiden entscheiden. Die jeweils nicht ausgewählte Seite bleibt damit im Bild erhalten. Zum leichteren Verständnis folgt dazu ein Beispiel.

Bild (grüner Kasten) mit noch unveränderter Bildebene

Die Ebene wird um 90° im Uhrzeigersinn gedreht und passt nicht mehr ins Bild

Die Ebene wird rechts abgeschnitten und der linke Teil bleibt übrig

Die Ebene wird links abgeschnitten und der rechte Teil bleibt übrig

Abbildungen 2.11: Beispiel für das Abschneiden beim Drehen

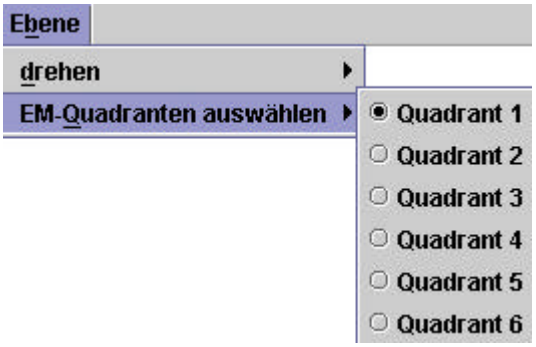


Abbildung 2.12: Menü zur Quadrantenauswahl

Bei Environment Maps wird eine Drehung oder Spiegelung immer nur auf einen Quadranten angewendet. Auf welchen Quadranten die Operation angewendet werden soll, kann man im Ebenenmenü unter `EM-Quadranten auswählen` bestimmen. Dabei sind die Quadranten in der Darstellung als Kreuz von links nach rechts von eins bis vier durchnummeriert. Der obere Quadrant ist Nummer fünf und der untere Nummer sechs.

2.7 Speichern

Abbildungen 2.13: Dateimenü bei geöffnetem Bild

Sowohl einzelne Bilder als auch Environment Maps lassen sich über das Menü `Datei > Bild speichern` im Portable Floatmap Format speichern. Ist nur ein einzelnes Bild geöffnet, wird dieses natürlich auch als eine einzelne Datei gespeichert. Bei Environment Maps wird jeder Quadrant als eine einzelne Datei gespeichert. Im Dialog gibt man dazu einen Dateinamen an, der vor der Endung um einen Bindestrich und die Nummer des Quadranten ergänzt wird. Gibt man beispielsweise `bild.pfm` als Dateiname für die Environment Map an, so werden die Dateien `bild-1.pfm` bis `bild-6.pfm` erzeugt. Die Angabe der Endung `.pfm` ist dabei optional und wird gegebenenfalls automatisch ergänzt. Existiert eine angegebene Datei bereits, so muss das Überschreiben per Dialog bestätigt werden.

2.8 Exportieren

Über das Menü `Datei > Bild exportieren` lässt sich ein gerade geöffnetes Bild im JPEG-Format abspeichern. JPEG ist ein Low Dynamic Range Format und die resultierende Datei hat nicht mehr die gesamte Farbinformation aus dem Bild. Vielmehr wird beim Export das Bild so, wie es gerade gammakorrigiert auf der Zeichenfläche angezeigt wird, abgespeichert. Der Export erfolgt ähnlich wie beim Speichern, so dass Einzelbilder auch in einer einzigen JPEG-Datei landen und Environment Maps aufgeteilt in die Quadranten in sechs JPEG-Dateien geschrieben werden. Dabei wird der angegebene Dateiname wieder um einen Bindestrich und die Nummer des Quadranten ergänzt.

3. Klassenbeschreibung

In diesem Kapitel werden die benutzten Klassen beschrieben. Alle Klassen sind in einer Package `hdr` enthalten.

3.1 Application

Die Klasse `Application` stellt den Einstiegspunkt für das Programm dar. Sie enthält die `main`-Methode und erstellt eine Instanz von `HDRFrame`.

3.2 HDRFrame

Das Malprogramm ist fensterbasiert und keine Konsolenanwendung, auch wenn gewisse Statusinformationen auf der Konsole ausgegeben werden. In der Klasse `HDRFrame` wird das Fenster implementiert, in dem die Benutzerschnittstelle dargestellt wird. Die wichtigste Funktionalität der Klasse ist das Erzeugen eines Menüs in Form der `HDRMenuBar` und des eigentlichen Fensterinhaltes, der durch ein `HDRMainPanel` dargestellt wird. Es gibt eine `start`-Methode, die die Oberfläche nach dem Programmstart initialisiert. Dieser muss ein Bild übergeben werden, welches im Menü durch Neuanlegen oder Laden erstellt werden kann. Die `stop`-Methode setzt die Oberfläche wieder auf den Anfangszustand direkt nach Programmstart zurück. Mit `doExit` wird der vom Programm genutzte Speicher wieder freigegeben und das Programm beendet. Daneben gibt es noch ein paar Hilfsmethoden.

3.3 HDRMenuBar, HDREbenenMenu

Die Klasse `HDRMenuBar` stellt die Menüleiste dar. Es gibt zwei Menüs: eines für Datei- und Programmoperationen und eines für Operationen auf den Ebenen. Das Ebenenmenü ist in der eigens dafür vorhanden Klasse `HDREbenenMenü` implementiert. Im Dateimenü kann man Bilder neu anlegen, laden, abspeichern, exportieren und schließen, sowie das Programm verlassen. Im Ebenenmenü kann man Dreh- und Spiegeloperationen auf Ebenen anwenden und bei Environment Maps auch festle-

gen, auf welchen Quadranten die Operation ausgeführt werden soll. Neben dem Konstruktor gibt es noch eine Methode `setMenuAvailability`, die – je nachdem, ob gerade ein Bild geöffnet ist oder eben nicht – festlegt, welche Menüpunkte man benutzen kann. Außerdem müssen noch Events verarbeitet werden. Dazu implementiert `HDRMenuBar` die abstrakten Methoden der Interfaces `ActionListener` und `MenuKeyListener`. Relevante Events werden an die Methode `processEvent` weitergebeben, die dann je nach gewähltem Menüpunkt die Events verarbeitet. Viele Menüpunkte sind auch direkt über Tastaturkürzel erreichbar. Eine Liste dazu gibt es im Anhang.

3.4 HDRMainPanel

Bis auf das Fenster an sich und das Menü sind alle Elemente der grafischen Benutzeroberfläche im `HDRMainPanel` enthalten. Die Oberfläche teilt sich in drei Bereiche auf: den Werkzeugkasten am linken Rand, die Informationsleiste am unteren Rand und die zentrale Anzeige des Bildes. Der Werkzeugkasten wird in der Klasse `HDRToolBox` erzeugt und enthält Funktionen zur Gammakorrektur, Farbwahl und zum Malen. Die Informationsleiste wird in der Klasse `HDRInfoPanel` realisiert und bietet Statusinformationen und Navigiermöglichkeiten. Das Bild wird im 2D-Modus durch das `HDRMalPanel` und im 3D-Modus durch das `HDR3DMalPanel` als zentrales Element der Oberfläche dargestellt. Nach dem Laden bzw. Neuerzeugen eines Bildes befindet sich das Programm im 2D-Modus, der durch die Methode `init2D` aufgerufen wird. Der 3D-Modus ist nur für Environment Maps da und kann in der Informationsleiste durch den entsprechenden Button eingeschaltet werden, welcher die Methode `switch2D3DModus` aufruft. Diese überprüft, ob das Bild eine Environment Map ist und schaltet dann durch den Aufruf von `init3D` in den 3D-Modus. In den 2D-Modus zurück geht es analog durch erneutes Aufrufen von `switch2D3DModus`.

Daneben speichert das Mainpanel das Bild, welches im nächsten Abschnitt näher beschrieben wird und stellt einige Getter-Methoden bereit, die den Zugriff auf das Bild und Teile der Oberfläche erlauben.

3.5 Bild, Ebene, Pixel, BildTools

Die `Pixel`-Klasse stellt ein einzelnes Bildpixel dar, welches den Rot-, Grün- und Blau-Anteil der Farbe des Pixels als Fließkommazahl speichert. Der eine Konstruktor erstellt aus drei Farbwerten ein neues Pixel, der andere kann durch Übergabe eines Pixels eine Kopie dieses Pixels erstellen. Ansonsten gibt es nur noch Getter und Setter, um die Pixelwerte zu erfahren bzw. zu manipulieren.

Eine `Ebene` fasst eine Menge von Pixeln in einem zweidimensionalen Array zusammen. Diese zwei Dimensionen entsprechen der Breite und Höhe der Ebene und des Bildes, falls es sich um ein einzelnes Bild handelt. Bei Environment Maps werden die Pixel aller sechs Quadranten ebenfalls in diesem Array gespeichert. Da die Quadranten quadratisch sind, wird die Breite und Höhe der Quadranten durch die Höhe des Arrays dargestellt. Die Breite des Arrays entspricht dem sechsfachen dieses Wertes, da die Quadranten in dem Array nebeneinander gespeichert werden. Die Klasse bietet auch noch Möglichkeiten, für jede Ebene einen Namen zu wählen. Es gibt einen Alphakanal, der die Opazität in Prozent angibt, und außerdem lässt sich die Sichtbarkeit der Ebene ein- und ausschalten. Allerdings werden diese Möglichkeiten vom Programm noch nicht weiter genutzt. Um diese ganzen Werte zu beeinflussen und herauszubekommen, gibt es natürlich Setter- und Gettermethoden.

Die Klasse `Bild` stellt nun das gesamte Bild aus Ebenen und Pixeln dar. Die Klasse ist etwas unübersichtlich geworden, da hier neben dem eigentlichen Bild auch noch weitere Informationen gespeichert werden. Daher sind alle statischen Methoden, um etwas Übersicht zu schaffen, in die Klasse `BildTools` ausgelagert. Eine Instanz von `Bild` generiert man, indem man zunächst eine Ebene erstellt und diese dann an den Konstruktor übergibt. Die Ebenen werden in einem Array gespeichert, wobei zu beachten ist, dass die Ebene mit dem niedrigsten Index in dem Array am weitesten unten und die mit dem höchsten am weitesten oben liegt. Somit kann also eine höherindizierte Ebene darunter liegende verdecken. Da die Pixel High Dynamic Range Werte enthalten, die auf dem Monitor nicht direkt darstellbar sind, wird im Bild für den 2D-Modus ein darstellbares Bild erzeugt, welches im Weiteren Javabild genannt wird.

Der Prozess, aus dem High Dynamic Range Bild ein Low Dynamic Range Javabild zu erzeugen, heißt Tonemapping und ist hier durch einen Algorithmus zur Gammakorrektur realisiert. Dazu wird zunächst der Maximalwert m in den RGB-Kanälen aller Pixel im Bild gesucht. Dieser entspricht später dem maximalen Farbwert von 255. Die HDR-Werte werden im ersten Schritt mit dem Maximalwert auf $[0;1]$ normalisiert.

Daraufhin erfolgt die eigentliche Gammakorrektur:

$$L_{normalisiert} = H_{normalisiert}^{\frac{1}{g}}$$

Dabei ist $L_{normalisiert}$ der auf $[0;1]$ normalisierte LDR-Wert, $H_{normalisiert}$ der auf $[0;1]$ normalisierte HDR-Wert und g der Gammawert. Der LDR-Wert muss noch mit 255 multipliziert werden, um ihn auf $[0;255]$ zu bringen. Der gesamte Algorithmus lautet dann:

$$L = \left(\frac{H}{m}\right)^{\frac{1}{g}} \cdot 255$$

wobei L und H dann die LDR- bzw. HDR-Werte ohne Normalisierung sind. Der Gammawert g kann wie folgt gewählt werden:

$0 < g < 1$	Das Bild wird abgedunkelt.
$g = 1$	Es findet eine lineare Skalierung statt.
$g > 1$	Das Bild wird aufgehellt.

In Java ist das wie folgt realisiert.

Klasse BildTools

```
public static int computeARGB(Pixel pixel, int alpha,
     float max, float eins_durch_gamma) {
     int a = alpha * 255 / 100;
     int r = (int) (Math.pow(pixel.getR() / max,
          eins_durch_gamma) * 255);
     int g = (int) (Math.pow(pixel.getG() / max,
          eins_durch_gamma) * 255);
     int b = (int) (Math.pow(pixel.getB() / max,
          eins_durch_gamma) * 255);
     return ((a << 24) | (r << 16) | (g << 8) | b);
}
```

Übergeben werden muss ein Pixel, welches die HDR-Werte für den Rot-, Grün- und Blau-Kanal enthält, der Alphawert für die Opazität, der Maximalwert im Bild und der schon in die -1te Potenz erhobene Gammawert. Der Alphawert ist in Prozent angegeben und wird nur skaliert. Für die einzelnen Farbkanäle wird die Gammakorrektur wie oben beschrieben mit Normalisierungen und Skalierungen durchgeführt. Zurückgegeben wird ein Integer, dessen vier Bytes den Alpha-, Rot-, Grün und Blaukanal enthalten.

Das Javabild wird in der Methode `createJavaImage` erzeugt. Diese unterscheidet wieder zwischen einem einzelnen Bild und einer Environment Map. Im ersteren Fall ist das Ausmaß des Javabildes identisch mit denen des Bildes. Im zweiten Fall ist das Javabild viermal so breit und dreimal so hoch wie ein Quadrant. Die Quadranten werden im Javabild als ein aufgeklappter Würfel in Kreuzform dargestellt, wie auf der Abbildung 3.1 erkennbar ist. An den leeren Stellen werden einfach transparente Pixel im Javabild gesetzt. Damit das Javabild nicht bei jeder kleinen Veränderung neu berechnet werden muss, kann man auf die Methode zur Erzeugung des Javabildes nicht direkt zugreifen. Vielmehr muss man sich der Methode `getOutputImage` bedienen. Diese liefert das Javabild, überprüft dabei aber anhand eines Booleans, ob die Neuberechnung überhaupt nötig ist. Im 2D-Malmodus und beim Malen mit dem Stift werden zwar beispielsweise neue Pixelwerte in eine Bildebene geschrieben und auch in dem angesprochenen Boolean wird kenntlich gemacht, dass das Bild verändert wurde. Das Javabild selbst wird aber nicht neu angefordert, sondern im `HDRMalPanel` an den richtigen Stellen mit der per Tonemapping erzeugten Farbe in der Zeichenfläche übermalt. Erst wenn dann, um das Beispiel fortzuführen, der Gammawert verändert wird, ist eine vollständige Neuberechnung nötig.

Abbildung 3.1: Darstellung einer Environment Map im 2D-Modus

Für den 3D-Malmodus funktioniert das ähnlich, nur dass hier eine andere Art von Bildern, aus denen dann Texturen für den Würfel generiert werden können, gebraucht werden. Diese Bilder werden mit der Methode `getBuffImage` erzeugt und müssen als Seitenlänge eine Potenz von zwei haben, da dies eine Anforderung für Texturen in Java3D ist. Da das Javabild bereits berechnet ist, werden diese Bilder erzeugt, indem Ausschnitte aus dem Javabild, die den Quadranten entsprechen, in das zugehö-

rige Bild für die Textur gezeichnet werden. Der folgende Quelltextausschnitt, macht dies deutlich.

Klasse Bild, Methode `getBuffImage`

```
BufferedImage bi = new BufferedImage(textureSize, textureSize,
     BufferedImage.TYPE_INT_ARGB);
Graphics2D g = bi.createGraphics();
int l=-1, o=-1, r=-1, u=-1;
switch (n) {
     case 0 :  l = 0;         o = height;
               r = height;    u = 2 * height;
               break;
     // ... die Fälle für die anderen Quadranten
}
g.drawImage(img, 0, 0, textureSize, textureSize,
     l, o, r, u, null);
```

Zunächst wird ein `BufferedImage` angelegt, welches als Vorlage für die Textur einer Würfelseite dienen soll. Die Größe `textureSize` ist die kleinste Potenz von zwei, die größer als die Seite eines Quadranten ist. Die Variablen `l`, `o`, `r` und `u` bezeichnen die linke obere und die rechte untere Ecke im Javabild, welche ein Rechteck um einen der Quadranten (hier „`case 0`“, also der erste Quadrant) bilden. In das `Graphics2D`-Objekt des `BufferedImage` wird nun der Ausschnitt aus dem Javabild gezeichnet. Dabei findet eine Skalierung statt, da der Ausschnitt aus dem Javabild normalerweise etwas kleiner als die Texturvorlage ist.

Die weiteren Methoden der Klasse `Bild` sind hauptsächlich Getter und Setter und werden gegebenenfalls an anderer Stelle erläutert. Die Klasse `BildTools` enthält aber noch ein paar erwähnenswerte Methoden. Mit `createEMBild` wird aus sechs gleich großen, quadratischen Ebenen, die je für eine Würfelseite stehen, eine `Ebene` erzeugt, aus der wiederum ein `Bild` erzeugt wird. Die Methode `reduceEbenen` reduziert alle vorhandenen Ebenen eines übergebenen Arrays auf eine. Dabei wird für jede Pixelposition das Pixel aus derjenigen Ebene genommen, die am weitesten oben liegt (den höchsten Index im Array hat), das Pixel überhaupt gesetzt hat (darf nicht `null`, also transparent sein) und dabei sichtbar ist. Die weiteren Methoden sind `computeARGB`, welche schon bei der Beschreibung der Gammakorrektur vorgestellt wurde, zwei Methoden zum Drehen und Spiegeln und Methoden, die für Testzwecke Ebenen mit bestimmten Eigenschaften erzeugen.

3.6 HDRToolBox, MySlider

Die `HDRToolBox` besteht aus zwei Reitern. Der erste enthält Möglichkeiten zur Gammakorrektur und Farbwahl, der zweite die Malwerkzeuge. Im Konstruktor werden alle Oberflächenelemente erzeugt, initialisiert und gezeichnet. Es werden nur Standardswingelemente verwendet. Lediglich für die Slider wurde die Klasse `MySlider` entworfen, die dafür sorgt, dass die Slider eine feste Breite haben. Zur Eventverarbeitung werden die Interfaces `ActionListener` und `ChangeListener` implementiert. Letzterer reagiert mit der `stateChanged`-Methode auf die Slider, ersterer mit `actionPerformed` auf alle anderen anfallenden Events.

Wird der Farbwahlbutton gedrückt, werden die drei Texte aus den nebenstehenden Eingabefeldern eingelesen und versucht, diese jeweils in einen Fließkommazahlwert zu wandeln. Gelingt dies nicht, weil beispielsweise gar keine Zahl in einem Feld steht, oder ist eine Zahl negativ, so wird man per Dialog auf die Fehleingabe hingewiesen. Anderenfalls wird aus den drei Werten eine neue Farbe berechnet. Ist dieser Farbwert größer als das bisherige Maximum im Bild, so wird vermerkt, dass das Bild nach Hinzufügen dieses Farbwertes neu für die Ausgabe berechnet werden muss, da sich die gammakorrigierten Farbwerte nun aufgrund eines neuen Maximums geändert haben. Außerdem stimmt im diesem Fall die Farbvorschau nicht mehr mit den Farben im Bild überein, da hier verschiedene Maximalwerte als Grundlage der Gammakorrektur verwendet wurden. Daneben werden auch noch die Werte der Slider aktualisiert, so dass diese auch die aktuell gewählte Farbe anzeigen. Wird ein Slider benutzt, so geschieht nichts weiter, als dass der Wert in einem Eingabefeld anhand zweier zugehöriger Slider neu ausgerechnet und, um diesen Wert auch darzustellen, der Farbwahlbutton gedrückt wird. Für jeden Farbkanal gibt es zwei Slider. Einer gibt eine Größenordnung als Potenz von zehn an und der andere, wie viel Prozent dieses Wertes für die Farbe genommen werden sollen. Der Farbwert berechnet sich dann als $\text{neuer Farbwert} = \frac{\text{Prozentwert}}{100} \cdot 10^{\text{Exponent}}$. Zu beachten ist noch, dass die direkte Eingabe von Farben genauere Abstufungen bei den Farbwerten zulässt, da die Slider für die Prozentangaben nur eine Genauigkeit von 0,01 haben.

Die gegenseitige Aktualisierung von Slidern und Eingabefeldern kann natürlich zu ungewollten Rückkopplungen führen. Wenn also beispielsweise ein neuer Farbwert eingegeben, dieser in den Slidern übernommen wird und diese, aufgrund einer Änderung des Wertes, ihrerseits wieder die Textfelder aktualisieren. Um dies zu verhin-

dern, gibt es einen Boolean, in dem vermerkt wird, wer die Quelle der Änderung ist und der dafür sorgt, dass es zu keinen ungewollten Rückkopplungen kommt.

Benutzt man den Gammabutton, so wird versucht, aus dem zugehörigen Eingabefeld eine Fließkommazahl zu erstellen. Diese muss positiv und ungleich Null sein. Bei Falscheingaben wird man wieder per Dialog auf selbige hingewiesen. Ansonsten wird der neue Gammawert im Bild vermerkt und das Neuzeichnen des Bildes sichergestellt. Daneben wird auch der Farbvorschaubutton, dessen Farbe ja auch vom Gammawert abhängig ist, aktualisiert.

Wählt man eines der Werkzeuge, also den Stift oder die Farbpipette, aus, so wird die Farbe des zugehörigen Buttons rot, um die Auswahl kenntlich zu machen. Die Auswahl wird auch im Bild vermerkt. Beim Stift kann man noch die Größe der Malfläche angeben, die ebenfalls im Bild vermerkt wird. Die Anwendung der Werkzeuge ist dann in den Klassen zu den Malflächen (Kapitel 3.8 bzw. 3.9) näher beschrieben.

3.7 HDRInfoPanel

Die Informationsleiste, welche in der Klasse `HDRInfoPanel` realisiert ist, unterscheidet sich in ihrer Erscheinung davon, ob gerade der 2D- oder der 3D-Modus aktiv ist. Bei der Konstruktion werden die meisten Elemente der Oberfläche initialisiert und der Listener, den die Klasse selbst implementiert, zugewiesen. Zur Darstellung der Oberfläche muss erst die Methode `setBild` aufgerufen und dabei das Bild übergeben werden. `HDRInfoPanel` ist von dem Container `JPanel` abgleitet und so werden durch `setBild` zunächst einmal alle aus möglichen früheren Aufrufen noch vorhandene Oberflächenelemente entfernt. Danach werden die für den gerade aktiven Modus nötigen Elemente wieder eingefügt.

Im 2D-Modus wird angezeigt, wie groß das Bild ist. Bei einem einzelnen Bild wird die Breite und Höhe dieses Bildes angezeigt, bei einer Environment Map die Breite und Höhe jedes Quadranten. Befindet man sich mit der Maus in der Zeichenfläche, wird unterhalb dieser Angabe die Position des Mauszeigers in Werten angegeben. Basis für diese Werte ist der Koordinatenursprung links oben in der Zeichenfläche. Beachten sollte man dabei, dass die Bilder von einem ein Pixel breiten Rahmen umgeben sind und das Pixel ganz oben links die Koordinaten x=0, y=0 hat. Daneben gibt es

noch einen Button, mit dem man in den 3D-Modus wechseln kann, welcher aber nur bei geöffneter Environment Map nutzbar ist.

Ist der 3D-Modus aktiv, so gibt es weiterhin die Anzeige für die Bildgröße und den Button, der zwischen 2D- und 3D-Modus umschaltet. Daneben gibt es eine Checkbox, mit der die Koordinatenachsen des lokalen Koordinatensystems des Würfels anzeigen kann und Buttons, um im 3D-Modus zu navigieren.

3.8 HDRMalPanel

Im `HDRMalPanel` ist die Zeichen- und Anzeigefläche für den 2D-Malmodus realisiert. Die Klasse erweitert `JPanel` und implementiert die Interfaces `MouseListener` und `MouseMotionListener`, um auf Eingaben der Maus reagieren zu können. Zur Anzeige von Bildern wird die Methode `paintComponent` überschrieben. In einem ersten Schritt wird überprüft, ob die vorhandene Referenz auf ein Bild auch wirklich auf ein konkretes `Bild`-Objekt zeigt. Dies ist bei der Konstruktion noch nicht der Fall, aber später durch den restlichen Programmablauf immer sichergestellt. Daher wird man auch nie eine weiße Zeichenfläche mit schwarzer Umrahmung und Kreuz sehen können, die insbesondere für Testzwecke noch vorhanden ist und bei nicht vorhandenem `Bild`-Objekt angezeigt wird. Stattdessen wird die Zeichenfläche zunächst anhand der im Bild gespeicherten Hintergrundfarbe übermalt, welche standardmäßig auf reines, vollgesättigtes Grün eingestellt ist. Als nächstes wird ein Rahmen um den Bereich gemalt, in den später das Bild gezeichnet wird. Der Rahmen ist immer einen Pixel breit und schwarz. Bei einem Einzelbild besteht der Rahmen aus einem Viereck, welches zwei Pixel größer als das zu zeichnende Bild ist. Das Bild selbst wird dann innerhalb dieses Rahmens gemalt. Bei einer Environment Map besteht der Rahmen aus zwölf Linien, die zusammen eine Kreuzform um die Quadranten bilden. Genau genommen wird nun das Bild mit der Environment Map über diesen Rahmen gezeichnet, aber aufgrund der transparenten Flächen an den Teilen des Bildes, die nicht zu den Quadranten gehören, bleibt der Rahmen weiter sichtbar.

Um Werkzeugoperationen auf der Zeichenfläche auszuführen, werden Mausaktionen mit `mouseClicked` und `mouseDragged` überwacht. Bei einzelnen Klicks auf die Zeichenfläche liefert `mouseClicked` die Position, an der geklickt wurde, an die `malen`-Methode. Bewegt man den Mauszeiger bei gedrückter Maustaste über die Zei-

chenfläche, so werden alle Positionen per `mouseDragged` an die `malen`-Methode weitergegeben.

Die `malen`-Methode überprüft zunächst, welches Werkzeug ausgewählt ist. Ist es der Stift, so wird als erstes das `Graphics`-Objekt des `HDRMalPanels` geholt, auf dem dann die Zeichenoperationen ausgeführt werden. Die aktuelle Low Dynamic Range Malfarbe ist im Bild vermerkt und wird im `Graphics`-Objekt eingestellt. Dann wird ein Bereich um die übergebene Position der Maus betrachtet, welche den Mittelpunkt eines Quadrates mit einer im Bild vermerkten Seitenlänge bildet. In diesem Quadrat wird die Zeichenfläche mit der erwähnten Malfarbe eingefärbt, wenn die einzelnen Positionen noch innerhalb des Bildes liegen. Parallel dazu werden im `Bild`-Objekt die Änderungen an den Pixeln vorgenommen. Dazu dient die `setPixel`-Methode von `Bild`, die die Pixel im Bild mit einer High Dynamic Range Farbe setzt, die der bereits erwähnten Low Dynamic Range Malfarbe entspricht. Zum Schluss wird noch im Bild vermerkt, dass eine Änderung stattgefunden hat. Das zweigleisige Malen hat zur Folge, dass die Zeichenfläche nicht bei jeder Änderung ständig neu gezeichnet werden muss.

Hat man statt dem Stift die Farbpipette gewählt, wird an der Mausposition die Farbe im Bild ermittelt und – falls man nicht gerade neben das Bild auf die Zeichenfläche geklickt hat – diese Farbe als aktuelle Malfarbe gesetzt.

Bewegt man die Maus über die Zeichenfläche, wird `mouseMoved` aufgerufen, welches die Position des Mauszeigers an die Informationsleiste liefert. Verlässt der Mauszeiger wieder die Zeichenfläche, so wird `mouseExited` aufgerufen und die letzte Positionsangabe von `mouseMoved` in der Informationsleiste wieder gelöscht.

3.9 HDR3DMalPanel, MyMouseZoom, MyMouseRotate

Im `HDR3DMalPanel` wird der gesamte 3D-Malmodus realisiert. Dazu wird wieder mit dieser Klasse der Container `JPanel` erweitert, in den als 3D-Zeichenfläche ein `Canvas3D` gelegt wird. Nun wird der erste Teil des Szenegraphen als `SimpleUniverse` erzeugt. Damit braucht man sich um viele Dinge nicht mehr zu kümmern und nur die Kamera wird noch etwas aus dem Koordinatenursprung wegbewegt, so dass dort dann der Würfel erzeugt werden kann. Im Konstruktor wird noch eine Transformationsmatrix für Bewegungen des Würfels und ein Teilbaum, der den Würfel enthält, er-

zeugt und ins Universum gehangen. Abbildung 3.2 zeigt, wie das `SimpleUniverse` aufgebaut ist und wo der Teilbaum („ContentBranch“) hineingehangen wird.

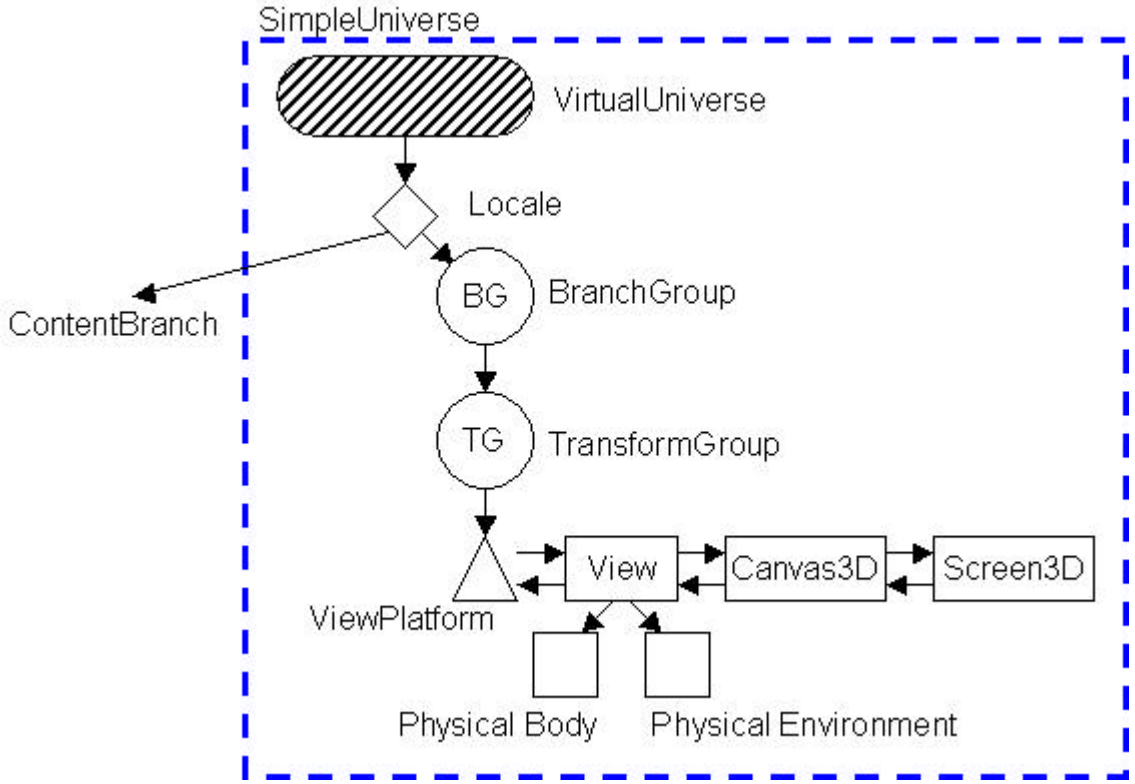


Abbildung 3.2: SimpleUniverse

Dieser Teilbaum wird mit der Methode `createContentBranch` erzeugt. Als obersten Knoten enthält er eine Transformationsgruppe, die wiederum die bereits erwähnte Transformationsmatrix für den gesamten Teilbaum enthält. An diese Gruppe werden nun der Würfel, ein Schalter und zwei so genannte Behaviours gehangen.

Der Würfel selbst ist als Gruppenknoten realisiert und wird mit der Methode `buildCube` erstellt. Er besteht aus sechs Seiten, die jeweils aus einem `Shape3D`-Objekt bestehen, welche wiederum aus Geometrie und Erscheinungsbild („Appearance“) zusammengesetzt sind. Die Geometrie wird aus `QuadArrays` erstellt. Jedes `QuadArray` besteht aus vieren der acht Eckpunkte des Würfels und hat für das Erscheinungsbild noch Texturkoordinaten gespeichert. Das Erscheinungsbild wird durch eine `Appearance` geprägt, welche in Abhängigkeit von der Würfelseite durch die Methode `createCubeAppearance` erzeugt wird. Diese legt fest, dass für die Polygone kein Culling stattfindet, sie also von beiden Seiten gerendert werden müssen und führt dazu, dass man auch in den Würfel hineinfliegen kann. Außerdem wird eine Textur erzeugt, mit der die angegebene Würfelseite überzogen wird.

Der Schalter wird in der Methode `buildSwitch` erzeugt. Er enthält einen Gruppenknoten, dessen Anzeige ein- und ausgeschaltet werden kann. Dieser Gruppenknoten besteht aus drei `Shape3D`s, die für die Koordinatenachsen des Koordinatensystem des Würfels stehen. Über Erscheinungsbilder sind sie verschieden gefärbt, um sie unterscheiden zu können.

Die zwei „Behaviours“ sind in den Klassen `MyMouseRotate` und `MyMouseZoom` implementiert. Sie sorgen dafür, dass man den Würfel mit der Maus drehen und heran- oder wegzoomen kann. Die beiden Klassen habe ich nicht selbst erstellt, sondern nur die Klassen `MouseRotate` und `MouseZoom` aus der Package `com.sun.j3d.utils.behaviors.mouse` genommen und diese nach Vorgaben von Eric Reiss[1] angepasst. Rotieren kann man den Würfel nun, indem man die mittlere Maustaste gedrückt hält und die Maus dabei bewegt. Der Würfel wird an der X- und Y-Achse des Weltkoordinatensystems rotiert. In der Informationsleiste hingegen wird an den drei Achsen des lokalen Koordinatensystems des Würfels rotiert. Das Zoomen funktioniert ähnlich, man muss hier die rechte Maustaste gedrückt halten und dann die Maus von sich weg oder zu sich hin bewegen.

Es gibt nun eine Reihe von Methoden, die verwendet werden können, um den Szenegraphen zu beeinflussen. Mit `addTransformation` kann man den Würfel mit einer Matrix beeinflussen. Genutzt wird das beispielsweise, um ihn zu rotieren. Die `addCameraMove`-Methode ist dafür da, die Kamera vor und zurück zu bewegen, um einen Zoomeffekt zu erzielen. `resetTransformation` macht Änderungen wieder rückgängig, indem in der `TransformGroup` des Teilbaums mit der Geometrie die Transformationsmatrix auf die Einheitsmatrix und damit wieder zurückgesetzt wird. Auch die Kamera wird damit wieder auf die Startposition gebracht. Mit `showInfos` kann man den Schalter benutzen und damit die Koordinatenachsen ein- und ausblenden. Die `update`-Methode dient zur Aktualisierung der Erscheinungsbilder der Würfelseiten. Übergibt man die Nummer eines Quadranten, so wird nur eine Würfelseite aktualisiert, ansonsten alle.

Klickt man irgendwo in die Zeichenfläche, bearbeitet die `mouseClicked`-Methode die Events. Wird dabei der Würfel getroffen, so wird in einem ersten Schritt ermittelt, welche Seite getroffen wurde. Die lokalen Koordinaten des Treffers werden an die Hilfsmethode `coordinates2ImagePosition` übergeben, welche daraus die Posi-

[1] siehe http://jamaica.ee.pitt.edu/Eric/java3d/behaviors/

tion im Bild ausrechnet. Mit dieser Position wird dann die `malen`-Methode aufgerufen, die dann erst mal nachsieht, ob der Stift oder die Pipette ausgewählt wurden. Ist der Stift ausgewählt, so wird analog zum 2D-Modus mit der Position als Mittelpunkt eines Quadrates mit im Bild vermerkter Seitenlänge eine Region, die dem Quadrat entspricht, mit der aktuellen Malfarbe gefärbt. Bei der Farbpipette wird einfach nur die an der entsprechenden Position vorhandene Farbe als aktuelle Malfarbe gesetzt.

3.10 PFMLoader, PFMSaver und das Portable Floatmap Dateiformat

Unter dem Namen Portable Floatmap (PFM) finden sich im Internet verschiedene Formate für High Dynamic Range Bilder. Ich benutze das auch von Paul Debevec in seinem HDR-Shop verwendete Format. Die Dateien haben die Endung „`.pfm`“ und sind in ihrem Aufbau an die Formate Portable Grayscale Map (PGM) bzw. Portable Pixmap (PPM) angelehnt. Letztere sind ASCII-Dateien, aber es macht natürlich keinen Sinn, Fließkommazahlen ASCII-kodiert abzuspeichern. Daher besteht nur der Dateiheader aus ASCII-Zeichen, die Farbinformationen sind binär kodiert.

Der Dateiheader enthält drei Einträge, die jeweils mit dem Newline-Zeichen (`0x0A`) abgeschlossen sind. Der erste Eintrag („magic number“) besteht aus den zwei Zeichen „PF“ und identifiziert die Datei als Portable Floatmap. Danach folgen die Breite und Höhe des Bildes, wobei die beiden Werte durch ein Leerzeichen (`0x20`) getrennt werden. Der dritte Eintrag lautet immer „-1.000000“ und über die Bedeutung kann ich nur spekulieren. Vermutlich rührt er von der bereits erwähnten Verwandtschaft zu den PGM-/PPM-Formaten her, aber er hat hier keine Bedeutung.

Nach dem letzten Newline-Zeichen des Dateiheaders folgen nun die eigentlichen Bilddaten. Diese sind nun binär kodiert. Jedes Bildpixel umfasst zwölf Bytes, die sich in je vier Bytes für den Rot-, Grün- und Blaukanal aufteilen. Diese vier Bytes stellen jeweils eine vier-Byte-Fließkommazahl nach IEEE-754[2] dar, die einem `float` in Java entspricht. Allerdings sind die Bytes little endian-kodiert, die am wenigsten signifikanten Bytes stehen also vorne. Um die big endian-Kodierung von Java zu erreichen, muss einfach die Reihenfolge der vier Bytes vertauscht werden. Dann steht an Bitposition 31 (ganz links) das Vorzeichen, die Bits 30-23 enthalten den Exponenten und die Bits 22-0 die Mantisse. Des Weiteren ist noch die Reihenfolge der Pixel zu

[2] siehe http://grouper.ieee.org/groups/754/

beachten. Das erste Pixel in der Datei entspricht dem Pixel links unten im Bild und das letzte Pixel in der Datei entspricht dem Pixel rechts oben im Bild. Anders formuliert stehen die Bildzeilen beginnend mit der untersten Zeile und endend mit der obersten in der Datei.

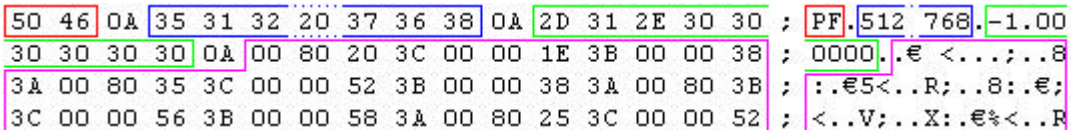


Abbildung 3.3: Ausschnitt einer im Hexeditor geöffneten PFM-Datei

Die Abbildung 3.3 zeigt einen Ausschnitt einer PFM-Datei in einem Hexeditor – links jeweils die Bytes in hexadezimaler Schreibweise und rechts entsprechend die ASCII-Interpretation. Rot markiert ist die Identifikation als PFM. Dann folgen Breite und Höhe (blau) – das Bild ist also 512 mal 768 Pixel groß – und der grün gerahmte dritte Headereintrag. Die Farbwerte sind magentafarben umrahmt.

Mit `PFMLoader` kann eine Portable Floatmap in eine `Ebene` geladen werden. Dazu wird die Methode `getPFM` aufgerufen und der Dateiname mit vollständigem Pfad übergeben. Die Implementierung ist recht fehlertolerant, weshalb man auch Dateien öffnen kann, die gar keine PFMs sind, was aber zu unvorhersehbaren Bildern führen kann. Wenn an dem übergebenen Pfad eine Datei existiert, wird zunächst der Header ausgelesen, der wie weiter oben beschrieben aufgebaut sein sollte. Mit der Breite und Höhe aus dem Header wird dann eine Ebene erzeugt, welche dann noch mit den Farbwerten aus der Datei gefüllt wird. Dies funktioniert für einen einzelnen Wert wie folgt.

Klasse PFMLoader, Methode `getPFM`

```
buff[0] = dataIn.readByte();
buff[1] = dataIn.readByte();
buff[2] = dataIn.readByte();
buff[3] = dataIn.readByte();
bits = (buff[3] << 24) |
       ((buff[2] << 16) & 0x00ff0000) |
       ((buff[1] << 8) & 0x0000ff00) |
       (buff[0] & 0x000000ff);
rgb[c] = Float.intBitsToFloat(bits);
```

`buff` ist ein Array aus vier `byte`, welches vom `DataInputStream dataIn` mit den vier Bytes einer Fließkommazahl gefüllt werden. Diese werden dann in umgekehrter

Reihenfolge in einen Integer geschrieben. Die bitweisen Und-Verknüpfungen sorgen dafür, dass es keine Fehler gibt, denn wenn die `byte`-Werte in ihrer Bedeutung als Zahl einen negativen Wert haben, überschreiben sie im Integer `bits` schon vorhandene Werte mit Einsen. `bits` enthält nun eine Bitfolge, die einem `float` entspricht. Das `float`-Array `rgb` steht für die in Rot-, Grün- und Blaukanal aufgeteilte Farbinformation eines Pixels und wird nun in einer Komponente mit der Farbe aus der Datei gefüllt.

Mit der `reset`-Methode kann man den `PFMLoader` wieder zurücksetzen und eine weitere Datei einlesen, ohne gleich eine neue Instanz erzeugen zu müssen. Die anderen beiden Methoden dienen der Fehlersuche.

Das Gegenstück zu `PFMLoader` ist `PFMSaver` und für das Speichern von Bildern zuständig. Der Methode `save` müssen die Dateinamen mit komplettem Pfad zum Speichern des auch zu übergebenden Bildes übergeben werden. Wenn es sich um keine Environment Map handelt, reicht ein Dateiname, ansonsten müssen es sechs sein, aus denen dann an den entsprechenden Stellen im Dateisystem Portable Floatmaps erzeugt werden. Beim Speichern wird zunächst wie folgt der Dateiheader geschrieben.

Klasse PFMSaver, Methode `save`

```
byte[] firstLine = {0x50, 0x46, 0x0A};
dataOut.write(firstLine);

int w = bild.getWidth();
int h = bild.getHeight();
if (bild.isEM()) w = h;
String width = Integer.toString(w);
for (int i = 0; i < width.length(); i++) {
     dataOut.writeByte(width.charAt(i));
}
dataOut.writeByte(0x20);
String height = Integer.toString(h);
for (int i = 0; i < height.length(); i++) {
     dataOut.writeByte(height.charAt(i));
}

byte[] thirdline = {0x0A, 0x2D, 0x31, 0x2E, 0x30, 0x30, 0x30,
     0x30, 0x30, 0x30, 0x0A};
dataOut.write(thirdline);
```

`dataOut` ist ein `DataOutputStream` mit dem man bequem einfache Datentypen in Dateien schreiben kann. Somit stehen „magic number“, Breite und Höhe, sowie die „-1.000000“ schon mal in der Datei. Nun folgen die Farbwerte.

Klasse PFMSaver, Methode `save`

```
rgb[0] = p.getR();
rgb[1] = p.getG();
rgb[2] = p.getB();
for (int i = 0; i < 3; i++) {
     temp = Float.floatToIntBits(rgb[i]);
     shift = (temp >> 24) & 0xff;
     b[3] = (byte) shift;
     shift = (temp >> 16) & 0xff;
     b[2] = (byte) shift;
     shift = (temp >> 8) & 0xff;
     b[1] = (byte) shift;
     shift = temp & 0xff;
     b[0] = (byte) shift;
     dataOut.write(b);
}
```

Alle Pixel werden in der bereits erwähnten Reihenfolge durchlaufen und die Farbwerte des aktuellen Pixels `p` in einem `float`-Array `rgb` gespeichert. Das Bitmuster jeder Fließkommazahl aus `rgb` wird in ein `int temp` übertragen. Dieses wird dann – aufgeteilt in sein vier Bytes und in umgekehrter Reihenfolge – in ein `byte`-Array `shift` gespeichert, welches wiederum in `dataOut` geschrieben wird. Transparente Stellen werden dabei im Bild später schwarz.

3.11 HDRNewDialog

Der Dialog, um ein neues Bild zu erstellen, wird durch die Klasse `HDRNewDialog` erzeugt. Der Dialog ist modal, man kann also, während er geöffnet ist, das Hauptfenster des Malprogramms nicht nutzen. Für ein Einzelbild muss man Breite und Höhe angeben. Bei Environment Maps reicht die Angabe der Breite, aber die entsprechende Option muss noch angekreuzt werden, wodurch auch das Eingabefeld für die Höhe verschwindet. Drückt man nun „Abbrechen“, verschwindet der Dialog wieder und es wird nur noch vermerkt, dass eben der „Abbrechen“-Button gedrückt wurde. Wird hingegen „Neues Bild erzeugen“ verwendet, so wird zunächst überprüft, ob gültige Werte eingegeben wurden. In den Eingabefeldern dürfen nur ganzzahlige Werte stehen, die größer als Null sind. Bei Falscheingaben erscheint ein Dialog, der auf den Fehler hinweist. Anderenfalls wird nach den Vorgaben aus den Eingaben ein Bild erzeugt Der Dialog verschwindet, und es wird noch vermerkt, dass der Button „Neues Bild erzeugen“ benutzt wurde. Mit der Methode `getStatus` kann man herausfinden, welcher der beiden Buttons gedrückt wurde und mit `getBild` kann man an das Bild – insofern es erzeugt wurde – gelangen.

3.12 HDROpenEMDialog

Auch `HDROpenEMDialog` ist ein modaler Dialog, er ist aber für das Öffnen einer Environment Map aus sechs gleich großen, quadratischen Portable Floatmaps da. In die Eingabefelder kann man entweder direkt den Dateinamen einer Portable Floatmap mit komplettem Pfad schreiben, oder man benutzt die nebenstehenden Buttons, um sich per Dialog zu einer Datei zu klicken. Hat man die sechs Felder ausgefüllt, muss man den „Laden“-Button drücken. Das Programm versucht nun, die sechs angegebenen Dateien zu öffnen und stellt dabei sicher, dass alle Dateien existieren, gleich groß und quadratisch sind. Sollte das nicht der Fall sein, so wird darauf mit einem Dialog mit Fehlermeldung hingewiesen. Hat alles geklappt, wird aus den sechs Dateien je eine Ebene und aus denen wiederum das Bild erzeugt. Man kann auch jederzeit den „Abbrechen“-Button benutzen – dann verschwindet der Dialog und es geschieht nichts weiter. Ähnlich wie beim `HDRNewDialog` gibt es auch hier die beiden Methoden `getStatus`, um herauszufinden, ob „Laden“ oder „Abbrechen“ gedrückt wurde, und `getBild`, um an das erzeugte Bild zu gelangen.

3.13 PFMFileChooser, JPEGFileChooser

`PFMFileChooser` und `JPEGFileChooser` erweitern die Klasse `FileFilter` aus der Package `javax.swing.filechooser`. Die beiden Klassen werden in Standarddialogen zum Öffnen und Speichern verwendet und ihre Ableitung hier führt dazu, dass nur Portable Floatmaps bzw. JPEG-Bilder in diesen Dialogen angezeigt werden. Es findet keine komplizierte Überprüfung der relevanten Dateien statt, sondern es wird lediglich nachgesehen, ob sie die Endung „`.pfm`“ bzw. „`.jpg`“ haben. Darüber hinaus gibt es noch eine Methode, die in den Dialogen eine kurze Beschreibung der PFMs oder JPEGs angibt.

3.14 VerticalFlowLayout

Die `VerticalFlowLayout`-Klasse stammt nicht von mir sondern von James Brundege. Sie stellt einen Layoutmanager dar, der ähnlich wie `FlowLayout` arbeitet. Container wie beispielsweise `JPanel`, die diese Klasse als Layoutmanager verwenden, richten alle Oberflächenkomponenten, die sie enthalten, in einer Spalte untereinander aus. Die einzige Änderung, die ich an der Klasse vorgenommen habe, war die Änderung der Package.

4. Installation

Dieses Kapitel beschreibt die Systemvoraussetzungen und wie man das Programm installiert und startet. Außerdem gibt es einige Hinweise, die beim Weiterentwickeln des Programms hilfreich sein könnten. Das Programm benötigt eine Fensterumgebung, sollte also auf jeder Windowsversion seit mindestens Windows 95, Mac OS und auch unter Linux bei beispielsweise installiertem KDE laufen.

Java

Zur Benutzung des Malprogramms muss ein Java-Interpreter installiert sein. Aktuelle Versionen gibt es für viele Betriebssysteme unter `http://java.com/de/`. Diese sind recht intuitiv zu installieren, weshalb hier nicht weiter darauf eingegangen werden soll. Sollte es Probleme geben, kann unter der genannten Internet-Adresse Hilfe gefunden werden. Das Programm wurde mit der Javaversion 1.4.2_01 erstellt, daher sollte darauf geachtet werden, diese oder eine neuere Version zu installieren.
Unter Windows muss man darauf achten, dass die Umgebungsvariable `path` den Pfad zur `java.exe` enthält, die den Java-Interpreter enthält. Ob das der Fall ist, kann man in der Eingabeaufforderung mit dem Befehl

```
echo %path%
```

überprüfen. Die Ausgabe sollte einen absoluten Pfad zu dem Ordner enthalten, in dem die `java.exe` liegt. Ist dies nicht der Fall, muss man der Umgebungsvariablen `path` den Pfad hinzufügen. Bei Windows XP geht dies, indem man in den Systemeigenschaften (Eigenschaften von Arbeitsplatz) unter dem Reiter „Erweitert“ auf den Button „Umgebungsvariablen“ klickt. Im neuen Fenster wählt man unter „Systemvariablen“ die Variable `path` aus, klickt auf „Bearbeiten“ und bei „Wert der Variablen“ fügt man am Ende des bisherigen Wertes zunächst ein Semikolon und dann den besagten Pfad zur `java.exe` ein. Schließt man alle Dialoge mit „OK“, sollte in einer neu geöffneten Eingabeaufforderung auch der Pfad mit ausgegeben werden.

Java3D

Für die Benutzung des 3D-Malmodus ist auch die Installation von Java3D nötig. Zunächst muss Java und danach Java3D installiert werden. Aktuelle Versionen sind unter `http://java.sun.com/products/java-media/3D/download.html` erhältlich. Das Programm wurde mit der Java3D-Version 1.3.1 erstellt. Es ist also diese oder eine neuere Version nötig.

Malprogramm

Das Malprogramm verfügt über keine Installationsroutine. Man kann einfach das Verzeichnis `Malprogramm` von der CD in ein beliebiges Verzeichnis auf der Festplatte kopieren. Für Windows ist eine Batch-Datei (`start.bat`) beigelegt, die man zum Starten des Programms verwenden kann. Auf anderen Plattformen wird das Programm gestartet, indem man in das Verzeichnis `Malprogramm` wechselt und dann in der Kommandozeile bzw. Shell den folgenden Befehl eingibt.

```
java –classpath . hdr/Application
```

Dabei ist „`java`“ der Java-Interpreter, „`-classpath .`“ gibt an, dass der Klassenpfad auf das aktuelle Verzeichnis (`Malprogramm`) zeigt und „`hdr/Application`“ ist die Java-Klasse `Application` aus dem Package (Unterverzeichnis) `hdr`, die gestartet werden soll.

Weiterentwicklung

Ich habe den JBuilderX mit JDK 1.4.2_01 und Java3D 1.3.1 genutzt, um das Programm zu erstellen. Die Projektdatei `HDR-Malprogramm.jpx` liegt im Verzeichnis `Quelltext` auf der CD.

Es folgen ein paar Hinweise zur Nutzung des JBuilders, die vielleicht auch bei anderen Entwicklungsumgebungen nützlich sind. Zunächst muss bei der Installation von Java3D darauf geachtet werden, dass in das JDK-Verzeichnis der Entwicklungsumgebung installiert wird. Beim JBuilderX unter Windows sieht das richtige Verzeichnis beispielsweise so aus:

```
C:\Programme\JBuilderX\jdk1.4\bin
```

Nach der Installation muss der JBuilder noch die Quellen aktualisieren. Das geht, in dem man im Menü „Tools“ den Punkt „Configure JDKs...“ wählt und dann bei „JDK home path“ auf „Change...“ klickt, dann aber nichts ändert, sondern den Dialog wieder mit „OK“ schließt. Der JBuilder katalogisiert danach alle .jar-Dateien neu, und Java3D sollte dann im JBuilder zu Verfügung stehen.

5. Anhang

5.1 Tastaturkürzel

Tastaturkürzel	Funktion	
Strg-E	Environment Map öffnen	stehen nur zu Verfügung, wenn noch kein Bild geöffnet ist
Strg-H	einzelnes High Dynamic Range Bild öffnen	
Strg-N	Neues Bild anlegen	
Strg-S	Bild speichern	stehen nur zu Verfügung, wenn schon ein Bild geöffnet ist
Strg-F4	Bild schließen	
Alt-F4	Programm schließen	

5.2. Glossar

Dynamic Range

Dynamikbereich: der Kontrast zwischen der hellsten und dunkelsten Stelle in einem Bild.

EM

siehe Environment Map

Environment Map

Zumeist für Beleuchtungssimulationen verwendete Textur aus High Dynamic Range Bildern. Vorstellbar als Würfel mit sechs Seiten, die hier auch als Quadranten bezeichnet werden.

Gammakorrektur

Im Malprogramm eingesetztes, sehr einfaches Tonemappingverfahren.

HDR

siehe High Dynamic Range

High Dynamic Range

Die Dynamic Range kann sehr groß sein und ist praktisch nur durch den Wertebereich der verwendeten Fließkommazahl begrenzt.

LDR

siehe Low Dynamic Range

Low Dynamic Range
: Die Dynamic Range in einem Bild ist stark begrenzt, da meist je Farbkanal nur 256 Abstufungen (1 Byte) möglich sind.

PFM
: siehe Portable Floatmap

Portable Floatmap
: Dateiformat für High Dynamic Range Bilder..

Quadrant
: Bezeichnet hier einen der sechs Teile einer Environment Map, die man sich auch als eine Würfelseite vorstellen kann.

Tonemapping
: Verfahren, um den großen Helligkeitsbereich von HDR-Bildern, der nicht von Monitoren darstellbar ist, so anzupassen, dass das Bild auf dem Monitor darstellbar ist. Aus dem HDR- wird zur Ausgabe also ein LDR-Bild erzeugt.

5.3 Inhalt der CD

Auf der beiliegenden CD sind die folgenden Inhalte in den angegebenen Verzeichnissen abgelegt.

`Beispielbilder`
: Einige Beispielbilder im Portable Floatmap Format, die alle von [1] stammen.

`Malprogramm`
: Das kompilierte Programm und eine Batchdatei (`start.bat`) für Windows, die das Programm ausführt. Außerdem eine Textdatei (`start.txt`) mit einer kurzen Installationsanweisung für andere Plattformen.

`Quelltext`
: Der Quelltext des Programms samt JBuilder-Projektdateien.

5.4 Referenzen

[1] Paul Debevec
http://www.debevec.org/
http://www.debevec.org/Research/HDR/
http://www.debevec.org/Probes/

[2] Billy Biggs, „Fun with high dynamic range imaging", Februar 2003
http://vektor.theorem.ca/graphics/tonemapping/tonemapping.pdf

[3] Erik Reinhard, „Tone mapping", University of Central Florida
http://www.cs.ucf.edu/~reinhard/classes/cap6938/tonemapping.pdf

[4] Informationen zu den PPM- und PGM-Dateiformaten
http://www.wotsit.org/

[5] Michael Höreth, „Tone-Mapping Verfahren zur Darstellung von High Dynamic Range Bildern", Studienarbeit an der Universität Koblenz-Landau, Juli 2003

[6] Java 3D API Tutorial
http://java.sun.com/products/java-media/3D/collateral/

[7] Christopher Schnell und Sascha Strasser, „Java3D – Ein Überblick der API"
http://www.student-zw.fh-kl.de/~chsc0013/downloads/Java3D.pdf