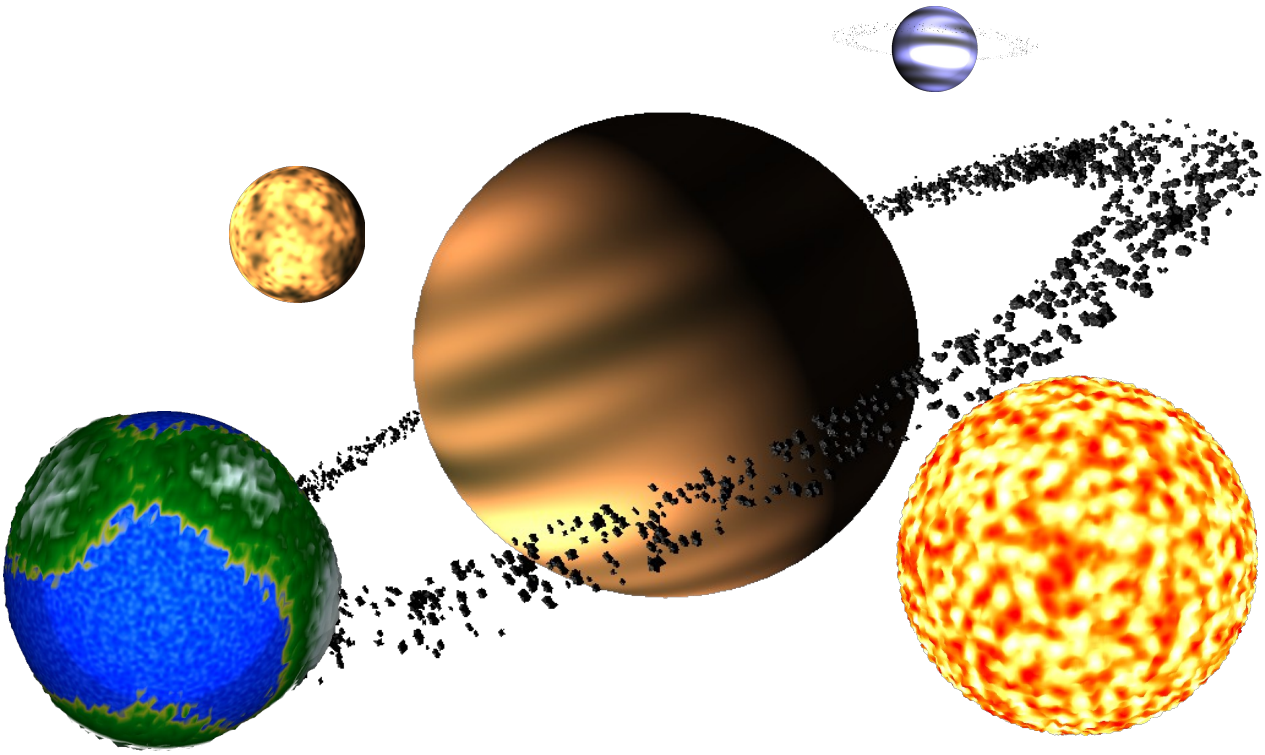
Florian Euchner, KS 1

Schuljahr 2013 / 14
Seminarkurs Informatik
Dokumentation

C++ / OpenGL / GLM / SOIL / OpenAL / Vorbis

Planether



Inhalt

1. Konzept / Zielsetzung

Der Hintergedanke bei dem Spiel „Planether“ ist es, die Faszination des Weltalls zu vermitteln. Durch Worte und Zahlen können wir die Abstände zwischen den Himmelskörpern, deren enorme Größe und die gewaltigen Relationen die schon innerhalb unseres eigenen Sonnensystems herrschen zwar beschreiben, allerdings kann sich kaum jemand angesichts der für uns unglaublichen Maßstäbe etwas unter diesen physikalischen Größen vorstellen.
Es gibt Modelle oder Bilder, die uns helfen, das All für uns begreifbarer darzustellen. Sie erlauben uns, die enormen Ausmaße von Sternen mit denen unseres Planeten zu vergleichen. Einrichtungen wie Astronomielehrpfade können die Abstände zwischen den Himmelskörpern darstellen. Doch trotz all dieser Vergleiche fehlt uns immer noch ein Gespür dafür, wie gigantisch weitläufig unsere nächste kosmische Umwelt ist. Außerdem können Bilder und Modelle zwar unsere Neugier befriedigen, idealerweise sollten sie aber auch ein Interesse wecken.

Der realitätsnahen Darstellung gegenüber steht eine ganze Reihe von Computerspielen, welche die Faszination für das unbekannte All aufgreifen und in kreativer Art verarbeiten. Schon sehr frühe Computerspiele wie „Lunar Lander“ oder „Asteroids“ (Atari, 1979) sind heute zu Klassikern geworden. Und irgendwie zieht sich bis heute die Weltraumthematik durch die Spielbranche. Getrieben von Science-Fiction Autoren, die schon immer ein Faible für die unendlichen Weiten hatten, entstanden und entstehen noch sehr viele Spiele, die das All als zentrales Spielelement betrachten. Bekannteste Vertreter davon ist „Star Cititzen“ (noch in Produktion). Hier wird logischerweise weniger auf die Realitätsnähe geachtet, stattdessen steht die Tauglichkeit als Spiel steht im Vordergrund. Dem gegenüber stehen Programme wie „Celestia“, die zwar wie Modelle und Grafiken die Relationen deutlich machen, aber nicht als Spiel herhalten können.

Ziel von Planether ist es, beide Welten, die der Spiele und die des Lernens, zu kombinieren. Daraus ergeben sich folgende zentrale Ziele:

- Spielspaß
  - Der Spielspaß steht, sofern keine großen Beeinträchtigungen existieren, über der Realitätsnähe. So sind im Spiel die Oberflächen der Planeten nicht originalgetreu, da das Aussehen der Planeten nicht im Vordergrund steht. Außerdem machen die Raketentriebwerke Geräusche, die eigentlich aufgrund des Vakuums im All natürlich nicht hörbar wären.
- Realitätsnähe
  - Die realistische Darstellung führt dazu, dass Spielern ganz nebenbei auffällt, wie klein ihr Raumschiff im Vergleich zur Erde ist und wie winzig es erst neben der Sonne erscheint. Absolut realistische Beschleunigungsverhalten von Raumschiffen und realistische Größen sind allerdings aufgrund des dadurch schwindenden Spielspaßes nicht implementiert,
  - Die Realitätsnähe wird aber vor allem bei physikalischen Größen wie die Ausmaße von Planeten, deren Abstände und der sich daraus ergebenden Umlaufbahnen streng eingehalten.
- Technische Umsetzung und Erweiterbarkeit
  - Um das Programm erweiterbar zu gestalten, muss das Grundgerüst dazu fähig sein, möglichst viele Ressourcen auf dem Computer zu verwenden.

Dazu zählt unter anderem die Multithreading-Unterstützung. Hierbei werden alle Aktionen von Objekten im Universum von allen CPU-Kernen gleichzeitig berechnet, was deutliche reale Performancevorteile mit sich bringt. Zudem wird möglichst viel Berechnung, z.B. die Terraingenerierung auf die GPU ausgelagert, da sich der Trend abzeichnen lässt, dass dem CPUs nicht mehr viel schneller werden, GPUs aber weiterentwickelt werden.
    - Ein dynamisches, objektorientiertes Grundgerüst führt dazu, dass bereits in wenigen Zeilen Quellcode ein weiterer Planet hinzugefügt werden kann. Während die Konstruktion dieses Fundaments anfangs viel Aufwand mit sich brachte, erleichtert es die schnelle und zuverlässige Erweiterung der Software. Dazu gehört auch, dass möglichst viele Aufgaben, wie die Beleuchtung oder die Physik, schon vom Spielkern abgenommen werden.
    - Portierbarkeit ist notwendig, da Planether auf Linux entwickelt wurde, aber auch auf Windows lauffähig sein soll. Dazu wird der MinGW-w64 Cross-Compiler verwendet.
- Freie Software
    - Freie Software bietet häufig den Vorteil, dass sie die Voraussetzung der Portierbarkeit unterstützt. Freie Bibliotheken sind dabei FreeGLUT, OpenGL, OpenAL oder auch libvorbis.
    - Der Grundsatz der freien Quellen gilt auch bei Texturen und Grafiken, die größtenteils unter Creative-Commons-Lizenzen stehen oder Public Domain sind.

2. Installation unter Windows

Zwar werden die Bibliotheken dynamisch gebunden, dennoch ist keine Installation erforderlich: Alle Bibliotheken sind als DLLs mit enthalten. Es muss lediglich die Datei „planether.exe“ im Ordner „win32“ ausgeführt werden. Es ist dabei ratsam, den Ordner erst von der DVD auf die Festplatte zu kopieren, da sich dadurch die Ladezeit verkürzt.
Unter Windows wird die Mauseingabe unzureichend unterstützt, deswegen kann hier das Numpad als Ersatz verwendet werden, sofern die num-Funktion aktiviert ist (nicht als Pfeiltasten). Die Tasten 2-8 stehen für oben-unten, 4-6 für links-rechts. Die Tasten 7 und 9 können zur Veränderung des Wankwinkels (später „Roll“ genannt) genutzt werden.
Getestet wurde das Spiel mit Windows XP und Windows 7.
Da GLUT unter verschiedenen Betriebssystemen verschiedene Eingabeevents für Maus und Tastatur liefert, entstehen in der Windows-Version noch einige Bugs, vor allem bei der Eingabe.

3. Kompilierung unter Linux

Abhängigkeiten für Planether sind die Bibliotheken OpenGL mit GLU, freeglut, glew, OpenAL, libogg, libvorbis sowie ein Compiler. Standardmäßig ist im Makefile clang++ voreingestellt.
Optionale Abhängigkeiten zur Entwicklung sind doxygen (zur Generierung der automatischen Quellcode-Dokumentation) sowie valgrind (zum Aufspüren von möglichen Bugs und Memory Leaks).
Zum erstellen des Programms muss im Terminal per in das Verzeichnis des Programms gewechselt werden. Die folgenden Befehle können zum Kompilieren verwendet werden.

```
make # Das Projekt erstellen
make run # Erstellen + ausführen
make cross # Für Windows erstellen, siehe 5.
make valgrind # Erstellen und mit valgrind ausführen
make -j12 # -j gibt die Anzahl der zur verwendenden Threads für
die Erstellung an; bei mehreren Prozessorkernen kann somit die
Dauer der Kompilierung verkürzt werden
```

4. Benutzung und Steuerung

Nach dem Starten der planether.exe bzw. unter Linux dem ausführen des Befehls „`make run`“ oder dem Starten der Verknüpfung öffnet sich das Spielfenster mit einem Splashscreen, solange das Programm noch Texturen, Shader und Sounds lädt. Der Ladevorgang kann in einem Terminal mitverfolgt werden.

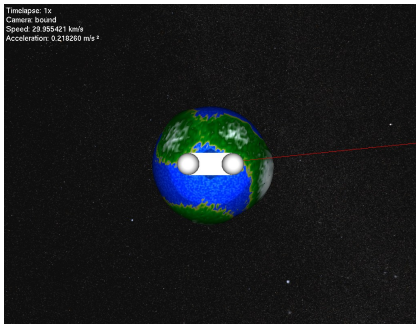


Abbildung 1: Anfangs befindet sich der Spieler Nahe der Erde

Bevor mit der eigentlichen Steuerung begonnen werden kann, ist es wichtig zu wissen, wie das Programm beendet werden kann, da die Maus vom Spiel gefangen wird. Die Maus lässt sich aus dem Spielfenster lösen, indem die Escape-Taste gedrückt wird. Zurück gelangt man in das Fenster wird einfach durch Klicken auf eine beliebige Stelle im Spielfenster. Um das Spiel ganz zu beenden, kann unabhängig vom verwendeten Betriebssystem und Fenstermanager die Q-Taste gedrückt werden.

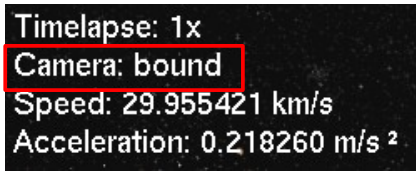
Mit F kann in den Vollbildmodus gewechselt werden, mit Shift-F wieder zurück in den Fenstermodus. All dies ist allerdings erst dann möglich, wenn das Spiel fertig geladen wurde.

Anzeigen und Verstecken von Elementen

Über die Tasten F1 und F2 können das Fadenkreuz / die Einblendungen (siehe Fehler: Referenz nicht gefunden) versteckt und wieder angezeigt werden. Über die Taste I kann zwischen einem Drahtgittermodell für die Planeten und plastischen Planeten umgeschaltet werden. Plastische Planeten sind standardmäßig eingestellt, sodass Drahtgitterplaneten in erster Linie zur Entwicklung und zum „hinter die Kulissen schauen“ gedacht sind.

Kamerabewegung

Neben dem Raumschiff ist auch die Kamera beweglich, sodass sich der Spieler leichter einen Überblick verschaffen kann. Der erste Schritt beim Erlernen der eigentlichen Steuerung sind die verschiedenen Kameramodi. Mit der Taste V können diese gewechselt werden.



In der Linken, oberen Ecke wird der aktuelle Kameramodus angezeigt

Verfügbar sind die Modi

- „bound“
  - Die Kamera befindet sich hinter der Raumschiff
  - Die Kamera kann weder bewegt noch gedreht werden
- „relative with rotation“
  - Die Kamera bewegt sich mit dem Raumschiff
  - Die Kamera kann gedreht werden; Eine Drehung des Raumschiffs dreht die Kamera immer gleich mit. Dadurch befindet sich das Raumschiff immer an

derselben Stelle auf dem Bildschirm

- „relative“
  - Die Kamera bewegt sich mit dem Raumschiff
  - Eine Drehung des Raumschiffes hat keine Auswirkung auf die Kamera
- „free“
  - Die Kamera bewegt und dreht sich unabhängig vom Raumschiff, bleibt in Relation mit der Sonne im Raum stehen

Anfangs ist der Modus „free“ besonders gut geeignet, um sich mit der Steuerung vertraut zu machen, dieser kann durch dreimaliges Drücken der v-Taste ausgewählt werden.

Um die Kamera zu bewegen, werden die Tasten WASD in der Standardkonfiguration verwendet. W beschreibt eine Bewegung in Blickrichtung, A eine Bewegung nach links, S eine Bewegung entgegen der Blickrichtung und D eine Bewegung nach rechts.

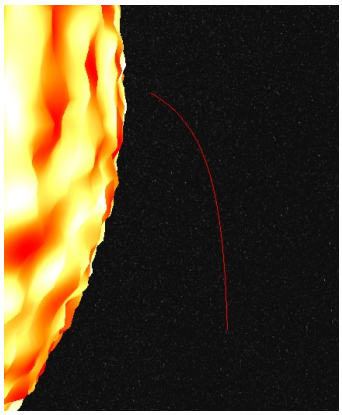


Abbildung 2: Gerade in der Nähe der Sonne lässt sich die Krümmung der Flugbahn durch die Gravitation beobachten

Außerdem können, wie in vielen Spielen üblich, die Leertaste zum Aufsteigen und die Taste E zum absteigen verwendet werden.

Die Drehung der Kamera wird mithilfe der Maus gesteuert. Der Roll-Winkel kann zudem durch Betätigung der linken oder rechten Maustaste variiert werden.

Zudem kann die Kamera mit speziellen Mäusen besonders komfortabel gesteuert werden. Hierbei dient das Mausrad für die schnelle Bewegung in und entgegen der Blickrichtung, die seitlichen Maustasten zur Bewegung nach oben oder nach unten und die Tasten nach rechts und nach links dienen der seitlichen Bewegung. Die Tasten nach rechts sowie nach links sind in der Regel im Mausrad integriert. Ein Klick auf die mittlere Maustaste beendet jegliche (seitliche Bewegung sowie die nach oben oder unten).

Microsoft Windows: Unter Windows kann es sein, dass die Steuerung mithilfe der Maus sehr träge reagiert. In dem Fall kann als Alternative die Steuerung mit dem Nummernblock verwendet werden. Siehe dazu [2. Installation unter Windows].

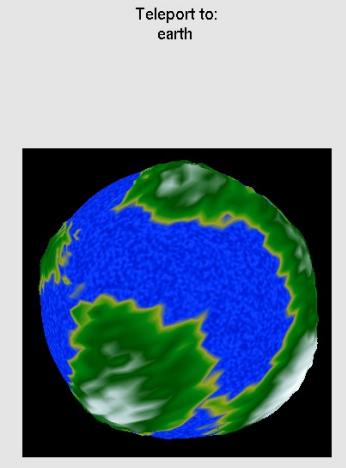


Abbildung 3: Hier wurde die Erde als Ziel der Teleportation ausgewählt

Raumschiffsteuerung

Das Raumschiff hat nur ein großes Raketentriebwerk, sodass der Beschleunigungsvektor immer abhängig von der Drehung des Schiffes ist. Da das Raumschiff physikalisch realistisch gehalten wurde, ändert es seine Bewegungsrichtung nicht durch eine Drehung, sondern nur durch Beschleunigung aufgrund der Gravitation oder aufgrund von Beschleunigung durch das Triebwerk. Zu beachten ist dabei, dass die Schubkraft des Triebwerks weit über realistischen Werten liegt, sie ist standardmäßig auf 100 $km\,s^{-2}$ eingestellt.
Drehungen des Raumschiffs werden mit den Pfeiltasten vollführt. Auch hier wird ein hoher Wert auf Realismus gelegt, sodass durch Betätigung der Pfeiltasten die Winkelgeschwindigkeit des Raumschiffs erhöht und herabgesetzt werden kann. Da es als Resultat besonders

schwierig wird, das Raumschiff in eine stabile Lage ohne Drehung zu versetzen, kann der Autopilot dies durch Betätigen der Taste C übernehmen.
Da es häufig erforderlich ist, das Raumschiff abzubremsen, hat auch dafür der Autopilot eine nützliche Funktion: Durch das Drücken das Bild Ab – Taste wird das Raumschiff so gedreht, dass das Raketentriebwerk genau in Richtung des Geschwindigkeitsvektors zeigt.

Das Raumschiff kann nun durch Drücken und Halten der Bild Auf – Taste nach vorn beschleunigt werden – analog zur Steuerung von Flugsimulatoren.
Als weitere Hilfe zur Steuerung dient die dünne, vom Raumschiff ausgehende rote Linie. Sie gibt zu jeder Zeit eine ungefähre, berechnete Voraussage der Flugbahn des Raumschiffs an. Zu beachten ist dabei, dass diese Flugbahn ausgehend von der aktuellen Position aller Himmelskörper und relativ zur Bewegung der Sonne angegeben wird.

Teleportieren

Da die Abstände zwischen den Planeten sehr groß sind und sich das Navigieren im Weltall vor allem für Anfänger ziemlich schwierig gestaltet, gibt es als einfache Alternative dazu das Teleportieren. Hierbei kann entweder das Raumschiff oder die Spielerkamera teleportiert werden. Um das Raumschiff an einen anderen Ort zu bewegen, muss mit der Taste T das Teleport-Fenster geöffnet werden. Hier lassen sich verschiedene Ziele (z.B. „saturn“ oder „sun“) auswählen und Vorschauansichten dieser Ziele betrachten. Durch die Pfeiltasten nach rechts und nach links lassen sich verschiedene Planeten und Sterne auswählen. Durch die Eingabetaste wird die Teleportation dann bestätigt, mit der Escapetaste kann das Fenster verlassen werden.

Zudem können Ziele, sofern deren Name schon bekannt ist, einfach über die Tastatur eingegeben werden.
Um statt der Spielerkamera das Raumschiff zu teleportieren, kann durch einmaligen Druck auf die Pfeiltaste nach oben die Option „Teleport Spaceship to“ ausgewählt werden.

Navigieren

Die Navigationsfunktion des Raumschiffes dient der einfacheren Wegfindung im Weltall. Aufgrund der großen Abstände ist es fast unmöglich, ohne Hilfe andere Planeten überhaupt zu finden.
Der Navigator wird aktiviert, indem man im Teleport-Fenster über die Pfeiltaste nach unten den Menüpunkt „Navigate to“ auswählt. Das Ziel der Navigation wird wie von der Teleportation gewohnt eingegeben.
Beim Navigieren wird eine weitere Linie eingeblendet, welche die Richtung vorgibt, in die das Schiff zum Erreichen des Ziels fliegen muss. Zudem erscheint ein Text, der die Strecke bis zum Ziel in Lichtjahren (ly), Lichtminuten (lm) oder Kilometern (km) angibt.

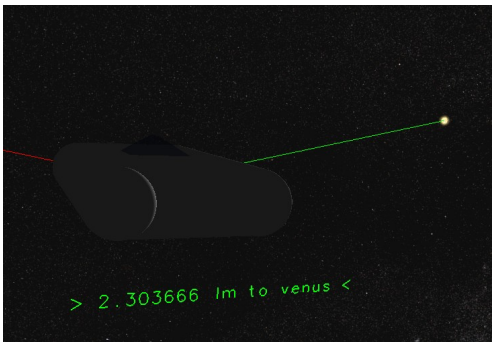


Abbildung 4: Das Raumschiff navigiert hier zur Venus

Durch das betätigen der Taste B richtet der Autopilot das Schiff in Richtung des Ziels aus.
Um die Navigation zu beenden, wird im Teleport-Fenster der Menüpunkt „Navigate to“ ausgewählt und ohne Eingabe mit der Enter-Taste bestätigt.

Veränderung des Seed

Zur zufälligen Generierung der Planetenoberflächen wird ein Seed für die Erzeugung der

Zufallswerte verwendet. Dieser kann durch die Tasten P und O erhöht oder erniedrigt werden. Dadurch kann zwischen verschiedenen Ansichten für die Planeten gewechselt werden.

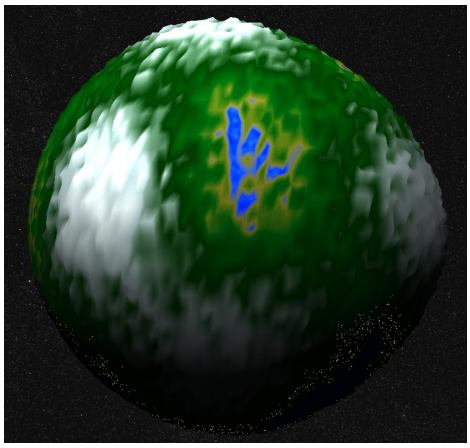


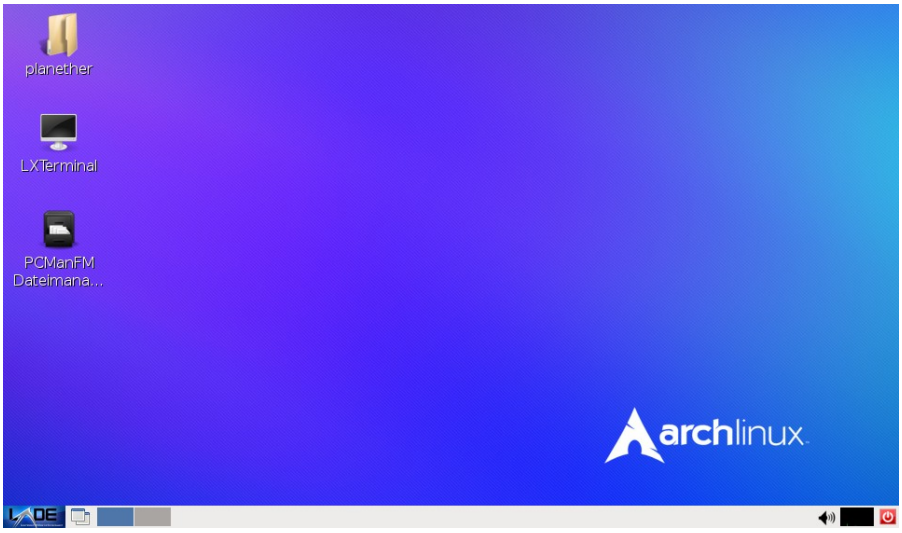
Abbildung 5: Mit einem anderen Seed kann die Erde sehr ungewohnt erscheinen

Spielziel

Das Spiel kann nicht „gewonnen“ werden, Ziel ist die Erkundung des Sonnensystems nach Belieben. Allerdings kann das Spiel verloren werden, sobald der Spieler mit dem Raumschiff in einen Planeten fliegt oder einem Stern zu nahe kommt. Der Spieler startet allerdings wieder nach einigen Sekunden in der Nähe der Erde. Dabei wird das Spiel vollständig zurückgesetzt.

5. Cross-Compilation für Microsoft Windows

Planether kann zum Erstellen einer ausführbaren Datei für Windows unter Linux Cross-kompiliert werden.
Hierbei wird der Cross-Compiler MinGW64 eingesetzt. Das Makefile erzeugt 32-bit Windows Executables, die auf verschiedenen Windows-Versionen laufen sollten. Alle benötigten Bibliotheken werden als DLLs mitgeliefert. Zudem übernimmt das Makefile auch die Bereitstellung der Texturen, Shader, Sounds in einen Ordner namens „win32“ im Planether-Ordner.
Zur einfacheren Verwendung mit Windows kann die mitgelieferte Virtual Machine Appliance „CompilerVM.ova“ in VirtualBox geladen werden. Hierzu wird in VirtualBox das Menü `Datei → Appliance importieren...` verwendet.
Vor dem ersten Boot sollte idealerweise schon ein gemeinsamer Ordner mit Schreibrechten eingerichtet werden. Dies geschieht durch einen Klick auf „Ändern“ während CompilerVM ausgewählt ist. Unter dem Menüreiter „Gemeinsame Ordner“ sollte hier der Ordner ausgewählt werden, in den später das Programm kopiert werden soll.
Nach dem ersten Start wird nach den Logindaten gefragt. Hierfür kann
Username: compiler
Passwort: compiler
verwendet werden. Nun wird ein minimalistischer LXDE-Dekstop angezeigt:



Um auf die Ordnerfreigabe mit dem Host-Windows zuzugreifen, muss der PCManFM

Dateimanager gestartet werden. In die Adressleiste wird nun zum Zugriff auf alle freigegebenen Ordner „/media“ eingegeben. Nun werden alle verfügbaren Freigaben angezeigt. Hier kann nun der gewünschte Ordner, der die zu kompilierende Planether-Version enthält, z.B. auf den Desktop kopiert werden. Im folgenden wird davon ausgegangen, dass sich ein Ordner namens „planether“ auf dem Dekstop befindet.

Um die Kompilierung vorzunehmen, wird nun per Doppelklick das LXTerminal gestartet. Über den Befehl

```
cd Dekstop/planether
```

kann im Terminal in den planether-Ordner auf dem Desktop gewechselt werden. Eine Eingabe von „ls“ (Ordnerinhalt anzeigen) sollte nun folgendes ergeben:

```
bin   Makefile   planether.desktop   sounds   textures
doc   obj        shaders             src      util
```

Sollte das nicht der Fall sein (u.U. können die Einträge „bin“ und „obj“ fehlen), wurde per „cd“ nicht in den richtigen Ordner gewechselt (Groß- / Kleinschreibung beachten).
Um das Programm für Linux zu kompilieren, muss nur der Befehl „make“ eingegeben werden. Nach erfolgreicher Erstellung kann Planether dann durch „make run“ gestartet werden. Um Planether für Windows zu kompilieren, muss der Befehl „make cross“ eingegeben werden.

```
[compiler@compilervm ~]$ cd Desktop/planether/
[compiler@compilervm planether]$ make cross
mkdir -p win32
i686-w64-mingw32-g++ -m32 -std=c++11 -pthread -w -DWIN32 -O3 src/spaceship.cpp s
rc/keyboard.cpp src/map.cpp src/drawutil.cpp src/audio.cpp src/player.cpp src/te
leport.cpp src/light.cpp src/gravity.cpp src/navigation.cpp src/tests.cpp src/sh
ader.cpp src/quatutil.cpp src/mouse.cpp src/environment.cpp src/skybox.cpp src/c
amera.cpp src/main.cpp src/hud.cpp src/debug.cpp src/splash.cpp src/objects.cpp
src/particle.cpp src/util.cpp src/SOIL/stb_image_aug.c src/SOIL/image_DXT.c src/
SOIL/SOIL.c src/SOIL/image_helper.c -o win32/planether.exe -lopengl32 -lglu32 -l
glut -lm -lglew32 -lOpenAL32 -lvorbisfile
```

Abbildung 6: Zwei Befehle reichen aus, um Planether für ein anderes Betriebssystem zu erstellen

Nachdem der Vorgang abgeschlossen ist, befindet sich die Windows-Version im Ordner „Desktop/planether/win32“. Dieser Ordner kann per Drag and Drop mit PCManFM in den von Windows freigegebenen Ordner gezogen werden.
Nach beenden der Virtuellen Maschine kann nun im „win32“-Ordner die „planether.exe“ ausgeführt werden. Das Spiel sollte sich nun öffnen.

6. Dokumentation von Algorithmen

Quaternionenkamera

Zur dreidimensionalen Drehung werden in der Regel roll-pitch-yaw-Winkel (Roll-Nick-Gier) verwendet. Dadurch werden allerdings Probleme wie unter anderem das „Gimbal Lock“ (http://de.wikipedia.org/wiki/Gimbal_Lock) erzeugt, die dazu führen, dass die klassischen Winkel nicht zur völlig freien Navigation im dreidimensionalen Raum verwendet werden

können.

Die sogenannten Hamiltonschen Quaternionen schaffen hier Abhilfe. Durch sie können Drehungen im Raum einfach kombiniert, subtrahiert und auf Vektoren sowie Matrizen angewendet werden. Die glm-Quaternionenklasse glm::quat enthält viele dafür notwendige Funktionen schon, wie das Kombinieren von Quaternionen mit dem *-Operator oder das Umwandeln von Quaternionen zu Rotationsmatrizen.

Die Verwendung der Quaternionen findet bei Planether allerdings nicht in erster Linie in der Kamera statt (die hauptsächlich der Darstellung von Objekten der Umgebung mit vorgegebenen Vektoren dient), sondern in der Player-Klasse. Diese nimmt Werte von Maus und Tastatur an, setzt sie in Bewegung um und erlaubt das Abrufen der Kameraachsen nach vorne, oben und rechts.

Dabei enthält die private Variable „m_look“ das Quaternion, das die Blickrichtung des Spielers beschreibt. Außerdem enthält die Variable „m_rotvel“ ein Quaternion, dass die Rotationsgeschwindigkeit und -richtung enthält. Die Umrechnung der Quaternionen in die entsprechenden Achsen erfolgt allerdings mithilfe der Funktion „`quatToMounting()`“ in quatutil.cpp.

```
void quatToMounting(glm::quat rotquat, SimpleVec3d *look, SimpleVec3d *up,
        SimpleVec3d *right)
{
        // Look Vector
        SimpleVec3d defaultLook(0, 0, -1);
        *look = rotateVecByQuat(defaultLook, rotquat);

        // Right Vector
        SimpleVec3d defRightAxis(1, 0, 0);
        *right = rotateVecByQuat(defRightAxis, rotquat);

        // Up Vector
        *up = crossProduct(*right, *look);
}
```

Der Funktion werden als Parameter das anzuwendende Quaternion sowie die zu beschreibenden Vektoren nach vorne, oben und rechts als SimpleVec3d-Pointer übergeben. Quaternionen sind dabei die relativen Drehungen der Koordinatenachsen nach vorne (0, 0, -1), rechts (1, 0, 0) und oben (0, 1, 0). Hier wird über die Funktion „rotateVecByQuat“ der entsprechende Vektor um das Quaternion gedreht. Der Vektor nach oben wird hierbei einfach über das Kreuzprodukt aus dem rechts und vorne-Vektor berechnet.

Eine weitere für die Quaternionenkamera relevante Funktion ist das Drehen der Kameraperspektive bei Bewegung der Maus oder bei der entsprechenden Tastatureingabe.

```
void Player::rotView(float xmov, float ymov)
{
        SimpleVec3d newlookdir(SimpleAngles(xmov, ymov, 0));
        SimpleVec3d defLookDir(0, 0, -1);
        m_look *= RotationBetweenVectors(defLookDir, newlookdir);
        m_look = glm::normalize(m_look);

        // Recalculate look, up, right axis
        quatToMounting(m_look, &m_lookaxis, &m_upaxis, &m_rightaxis);
```

```
}
```

Hier werden erst die yaw (xmov) und pitch (ymov) – Winkel in einen 3d-Winkel (SimpleAngles) eingesetzt. Daraus wird anschließend der Vektor berechnet, der den Winkel als Abweichung von der normalen Blickrichtung (0, 0, -1) darstellt. Diese Abweichung wird durch „RotationBetweenVectors“ in ein Quaternion umgerechnet, dass durch den „*=“-Operator zu m_look, dem Blickquaternion, hinzugefügt wird. Anschließend wird das Blickquaternion normalisiert und die Aufhängungsvektoren nach vorne, oben und rechts werden berechnet (s.o., quatToMounting).

Prozedural, parallel auf der Grafikkarte generierte Planeten

Keiner der Planeten, Sterne und Asteroiden in Planether wird mit einer Textur belegt. Stattdessen werden deren Farben und Formen jeden Frame neu auf der Grafikkarte mithilfe von GLSL-Shadern erzeugt. Dies ist heute aufgrund der stark optimierten und parallelisierten Grafikhardware möglich. Dadurch können beliebig viele verschiedene Planeten erzeugt werden. Außerdem kann Speicherplatz für Texturen eingespart werden. Lizenzierungsprobleme für Texturen werden zudem vermieden. Gegenüber einer Generierung auf der CPU bietet die Shader-Generierung den Vorteil, dass jederzeit und mit sofortiger Wirkung auf verschiedene Seeds umgeschaltet werden kann. Die CPU wird entlastet, während moderne Grafikkarten mit den Shadern kaum mehr Probleme haben. Allerdings werden die Vertices ursprünglich dennoch auf der CPU und nicht etwa auf der GPU generiert. Dadurch kann eine genauere Auflösung an den Stellen erreicht werden, an denen sich der Spieler befindet. Beim Annähern an Planeten werden also deren Gebirgszüge und Küstenlinien immer detaillierter (Erde, Mars, Mond, Merkur; nicht Planeten mit glatten Oberflächen wie z.B. die Gasplaneten).

Die Generierung der Vertices erfolgt mithilfe der SphereFraction-Klasse. Wie bei Quadtrees kann hier jede Instanz einer SphereFraction wiederum eine Reihe von Kinderelementen enthalten. Statt allerdings 2x2 Kinderelemente zu enthalten, stellten sich 4x4 Kinderelemente als performanter und besser aussehend heraus.
Eine detaillierte Dokumentation der SphereFraction-Klasse wäre zu umfangreich. Deswegen hier ein kurzer Überblick über die Funktionen der Klasse:

- `makePrototype(float radius, float dyaw, float dpitch)`: erzeugt eine Kugel mit radius und dyaw*dpitch Kinderelementen, nur nicht-Kind-Elemente
- `render()`: Zeichnet die vollständige Kugel auf den Bildschirm, nur nicht-Kind-Elemente
- `setChildren()`: Setzt die Anzahl der Kinderelemente, die die SphereFraction hat
- `updateVertices()`: Nur nicht-Kind-Elemente: Fragt rekursiv bei allen Kinderelementen nach allen Vertices der Kugel und setzt diese ins private Array m_quadvertices. Dieses wird dann in render() verwendet, um de Kugel auf dem Bildschirm darzustellen
- `autoChildrenNum(...)`: Setzt sich rekursiv durch alle Kinderelemente der SphereFraction fort. Diese Funktion wird dazu verwendet, um automatisch mehr Kinderelemente hinzuzufügen, wenn der Spieler in der Nähe ist oder um bei großer Entfernung wieder Elemente zu entfernen. Da dies einige Zeit dauern kann, sollte `autoChildrenNum(...)` und `updateVertices()` in einem seperaten Thread verwendet werden.
- `setChildrenStatic(...)`: Die Anzahl der Kinderelemente kann unveränderlich gemacht werden. Dies wird so z.B. bei den Gasplaneten oder bei Sternen

angewendet, die nicht mehr Vertices generieren sollen, wenn der Spieler sich Näher bei ihnen befindet.

Shader

Planether unterstützt sowohl Vertex- als auch Fragmentshader. Diese können unter anderem dafür verwendet werden, die Planeten zu generieren. Da Shader immer auf der Grafikkarte ausgeführt werden und für die spezielle Grafikhardware optimiert werden, müssen sie während des Programmstartes kompiliert werden. Der entsprechende Code hierfür befindet sich in shader.cpp im Konstruktor von ShaderManager.
Ein Shader namens builtin.glsl wird allgemein verwendet und in C++ einfach als String vorne in die Shaderquellcodes eingefügt. builtin.glsl enthält häufig verwendete Funktionen wie z.B. snoise3d(), eine Funktion die Simplex Noise im dreidimensionalen Raum generiert.
Mithilfe von Shader::loadShader() und Shader::loadShaderGroup() die jeweils Shader::importShader() verwenden, werden die einzelnen Shader in das Programm importiert und mithilfe von glCreateProgram, glCompileShader, glAttachShader, glLinkProgram zusammengefügt. Es werden in der Regel immer ein Vertex- und ein Fragmentshader in einer Gruppe zusammengefasst. Die Shadergruppen befinden sich im Unterordner „shaders“ und können dem Ordnernamen nach durch

```
game->getCamera()->getShaderManager()->requestShader(<ordnername>);
```

innerhalb des Programms jederzeit aufgerufen werden. Zudem werden einige häufig verwendete Werte wie die vergangene Zeit seit Programmstart den Shadern während der Laufzeit übergeben mithilfe der glUniform**()-Funktionen.
Benötigt ein bestimmter Shader zusätzliche Parameter, können diese mithilfe von

```
game->getCamera()->getShaderManager()->getShader(<shader>)->addParameter**(<name>,
<wert>);
```

dem Shader übergeben werden.

Multithreading

Die Taktraten heutiger Prozessoren verbessert sich zwar immer langsamer, dafür werden immer mehr CPU-Kerne hinzugefügt. Gerade um viele Objekte gleichzeitig zu berechnen bietet dies einen realen Vorteil. Ein von mir hochgeladenes Video auf YouTube soll den Unterschied demonstrieren: http://youtu.be/1O1FvP3BD0E

Die Anzahl der verwendeten CPU-Kerne kann in der Datei config.h mithilfe der Definition

```
#define CPU_CORES 6
```

geändert werden. Bei 6 CPU-Kernen wird die doppelte Zahl der Threads, also 12 verwendet.

Praktisch umgesetzt wird das Multithreading in der Datei environment.cpp . Die meisten Berechnungen finden in den ::step() - Funktionen der WorldObjects statt, darunter die Bewegung der Planeten, der FireParticle oder des Raumschiffs. Durch die Implementierung des Multithreading direkt im WorldEnvironment muss diese nur einmal geschehen und wird damit automatisch für alle Objekte angewendet.

Im Konstruktor von WorldEnvironment wird ein EnvironmentStepThreadManager() erzeugt:

```
m_threadman = new EnvironmentStepThreadManager();
```

Dieser startet in dessen Konstruktor die gewählte Anzahl an Threads:

```
EnvironmentStepThreadManager::EnvironmentStepThreadManager ()
{
	for (uint16_t i = 0; i < THREAD_NUM; ++i)
		m_threads.push_back(new EnvironmentStepThread());
}
```

Die Threads werden anfangs einmal gestartet statt bei jedem step() neu erzeugt zu werden. Dadurch wird eine Menge Zeit gespart, die das Starten und Beenden eines Threads benötigen würde. Trotz der Vorteile bedeutet dies, dass eine Kommunikation zwischen Threads nötig wird. Um diese kümmert sich die Klasse EnvironmentStepThread, die jeweils einen Thread umfasst. Sobald die Ausführung starten soll, werden dieser eine Liste der zu bearbeitenden Objekte (StepThreadObjects) übergeben. Da allerdings nicht nur ein Thread diese Objekte bearbeitet, enthält die Klasse StepThreadObject zudem einen Mutex.:

```
inline void StepThreadObject::tryExecute(float dtime)
{
	if (m_mutex.try_lock())
		m_object->step(dtime);
}
```

Die Funktion versucht bei der Ausführung eines Schrittes den Mutex zu sperren. Nur wenn die Funktion die Erste ist, die das versucht, ist es möglich. Ansonsten geht der Thread zum nächsten Objekt weiter und versucht dieses auszuführen.
Die Funktion, die dabei in einem anderen Thread abläuft nennt sich ::executor() ;

```
void EnvironmentStepThread::executor()
{
	while(m_running)
	{
		// Warten bis Haputthread das Startsignal gibt,
		// m_state wird auf 1 gesetzt
		// ansonsten noch eine Millisekunde warten
		while (m_state != 1)
			std::this_thread::sleep_for(std::chrono::milliseconds(1));

		...

		// Versuchen, die step()-Funktion des Objektes auszuführen
		// Die Funktion tryExecute
		// wird dies wie oben beschrieben nur dann tun,
		// wenn kein anderer Thread ihr zuvorkommt
		for (auto obj : *m_objects)
			obj->tryExecute(m_dtime);

		m_state = 0;
	}
}
```

Die Variable m_state ist dabei ein

```
std::atomic<bool> m_running;
```

damit keine Probleme mit zeitgleichem Zugriff aus verschiedenen Threads entstehen.

7. Dokumentation mit Doxygen

Zusätzlich zu dieser Dokumentation ist eine zweite, automatisch generierte, englischsprachige Dokumentation vorhanden. Diese bezieht sich vor allem auf den internen Aufbau des Programms, auf die Klassen, Funktionen und Parameter. Dabei werden auch z.B. automatisch Infografiken generiert, die Aufschluss über die Vererbung verschiedener Klassen voneinander u.v.m. bieten.

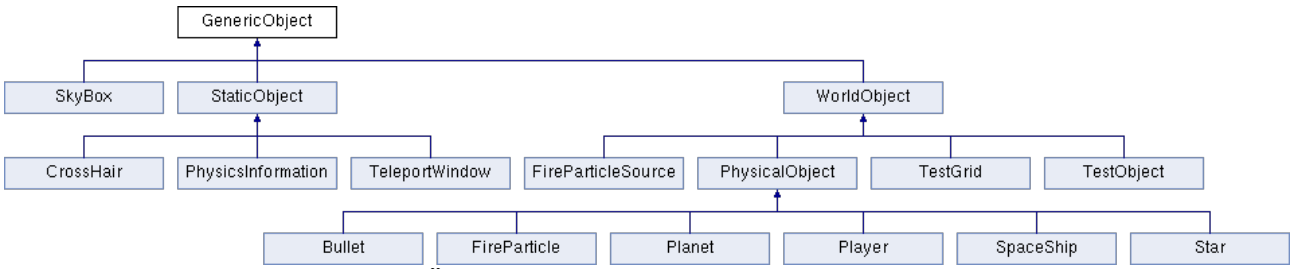


Abbildung 7: Doxygen gibt hier Übersicht über die von GenericObject abgeleiteten Klassen

Die Dokumentation wird durch das Programm Doxygen automatisch aus dem Quellcode generiert. Dafür sind im Quellcode teilweise speziell für Doxygen bestimmt Kommentare enthalten. Die Dokumentation kann unter Linux durch den Befehl
`make doxygen`
erstellt werden. Sie ist aber auch schon fertig auf der DVD mitgeliefert. Um die Dokumentation aufzurufen, muss im Ordner `doxygen` auf der DVD die Datei index.html mit einem Browser geöffnet werden.

8. Lizenzierung / Beitrag anderer

Bis auf die folgenden Ausnahmen wurde das Projekt von mir selbst erstellt:

- Alle Musik- und Sounddateien stammen aus den Open-Source-Spielen Scorched3d oder Warzone2100. Für Details siehe hierzu planether/sounds/LICENSE.txt auf der DVD.

- Die Texturen des Hintergrundbilds stammen von der NASA (http://svs.gsfc.nasa.gov/vis/a000000/a003500/a003572/). Siehe hierzu auch planether/textures/LICENSE.txt auf der DVD.

- Die Unterordner „glm“, „SOIL“ und „rapidjson“ in planether/src/ stammen aus drei anderen Softwareprojekten, nämlich
  - OpenGL Mathematics: http://glm.g-truc.net/ (Quaternionen-Hilfsfunktionen und Rotationsmatrizen)
  - Simple OpenGL Image Loader: http://lonesock.net/soil.html (Zum Laden von .png-Texturdateien)
  - Rapidjson: https://code.google.com/p/rapidjson/ (Zum Einlesen der config.json-Datei)

  - Alle drei Bibliotheken sind unter freien Lizenzen und können somit mit Planether weitergegeben werden

- Einzelne Codeausschnitte stammen aus anderen Softwareprojekten oder aus Anleitungen der entsprechenden Funktionen. Dies wird im Quellcode an den entsprechenden Stellen gekennzeichnet.

Anhang: Inhalte der DVD

- **planether:** Dieser Ordner enthält alle Quelldateien von Planether. Er ist der Projektordner von Planether.
  - **src**: Enthält alle CPP-Quelldateien und Headerdateien
  - **shader**: Enthält alle Shader (GLSL-Quelldateien)
  - **sounds** / **textures**: Enthält Musik und Klänge bzw. Texturen
  - **config**: Enthält die **config.json**-Datei, die für Einstellungen verwendet werden kann. Darin enthaltene Texte dienen zur Erklärung der Optionen.
- **win32:** Dieser Ordner enthält eine für Windows kompilierte Version von Planether. Getestet wurde diese unter Windows 7 und Windows XP

- **CompilerVM**: Siehe (5.), Virtuelle Maschine zur Erstellung von unter Windows ausführbaren Dateien
- **planether_dokumentation.odt**: Diese Dokumente in digitaler Form im odt-Format für LibreOffice Writer
- **planether_dokumentation.pdf**: Diese Dokumente im pdf-Format
- **autorun.inf / start_planether.bat / planether.ico**: Autostart-Datei, die dafür sorgt, dass beim einlegen der DVD automatisch Planether unter Windows gestartet werden kann sowie dazugehörige Dateien