# OpenGL unter Python


Department für Informatik

23.02.06

Dipl.-Inform.
Rüdiger Busch

# Inhaltsverzeichnis

# 1 Motivation

OpenGL hat sich zu einem Plattform übergreifenden Standard entwickelt und stellt aufgrund der Leistungsfähigkeit heutiger Grafikkarten eine verhältnismäßig einfache Möglichkeit dar, dreidimensionale Grafiken in Echtzeit zu präsentieren. Die Anwendungen reichen dabei von Spielen über Werkzeuge zur Grafikentwicklung bis hin zu komplexen CAD Systemen.

Im Zuge dieses Artikels soll der Einstieg in die OpenGL Programmierung gezeigt werden. Dabei sind allgemeine Kenntnisse der Computergrafik und der Programmiersprache Python zum Verständnis zwar von Vorteil, ein Nachvollziehen der angewendeten Techniken sollte aber aus der Kombination Quellcode + Artikel auch ohne besondere Vorkenntnisse möglich sein.

# 2 Python und OpenGL

Ursprünglich für C entwickelt stehen heutzutage OpenGL Anbindungen für fast alle gängigen Computersprachen zur Verfügung. Für den Einstieg in OpenGL, können verschiedene Kriterien herangezogen werden. Liegt der Schwerpunkt von vornherein auf einer hohen Ausführungsgeschwindigkeit, empfiehlt sich beispielsweise C++. Liegt der Schwerpunkt auf dem reinen Erlernen von OpenGL, ist es sinnvoll, eine Programmiersprache zu wählen, die leicht erlernbar und übersichtlich ist und nicht zu sehr von der eigentlichen Thematik ablenkt. Für diese Einführung wurde die Programmiersprache Python[python] gewählt, Gründe dafür sind im Folgenden aufgeführt.

## 2.1 Gründe für Python

- OpenSource und plattformunabhängig
  Python steht für die gängigen Betriebssysteme (Linux, Mac, Windows) kostenlos zur Verfügung. Ebenso gibt es gute frei verfügbare Entwicklungsumgebungen, z.B. Eric3[eric3], welche selbst mit Python entwickelt wurde.
- Klare, schnell erlernbare Sprachsyntax
  Gerade im Vergleich zu Perl oder C gibt es kaum kryptisch wirkende Befehlssequenzen
- Gut lesbarer Code
  Es fehlen BEGIN/END und Klammerkonstrukte, der Quelltext *muss* über sauberes Einrücken strukturiert werden.
- Viele Module vorhanden
  Für diverse Aufgabenfelder (Netzwerk, Grafik usw.) sind bereits umfangreiche Bibliotheken vorhanden.
- Skript-/Interpretersprache
  Über die Kommandozeile lassen sich kleinere Aufgaben schnell durchführen. Komplexe Anwendungen können leicht getestet werden, ohne erneute Übersetzungen durchzuführen. Dadurch wird effektives *rapid prototyping* unterstützt.
- Aussagekräftige und gezielte Fehlermeldungen
  Es gibt nicht nur eine klare Aussage, welcher Fehler aufgetreten ist, sondern einen *traceback*, der die Stationen des Fehlers bis zu seinem Ursprung zurückverfolgt.
- Objekte können *named parameter* verwenden
  Es können z.B. für Methoden Standardwerte vorgegeben werden, welche beim Aufruf der Methoden nicht mehr explizit aufgeführt werden müssen.
- Objektorientiert
  Python ist eine „echte“ objektorientierte Sprache
- Hohe Ausführgeschwindigkeit
  Obwohl Python interpretiert wird, lassen sich viele Berechnungen in Echtzeit durchführen, ohne dass es zu spürbaren Leistungseinbrüchen kommt.

## 2.2 Ziel dieser Arbeit

Das hauptsächliche Ziel dieser Arbeit ist es, einen einfachen Einstieg in die grafische Entwicklung mit OpenGL zu ermöglichen. Es wird das grundlegende Wissen vermittelt, wie eine virtuelle Welt unter OpenGL aufgebaut ist. Nach dem Durcharbeiten dieser Einführung sollte der Leser in der Lage sein, selbständig eine animierte Szene zu erschaffen, die dem in Abbildung 1 dargestellten Szenenausschnitt vom Komplextitätsgrad her entspricht.


Abbildung 1 Leuchtturm in Aktion

Als Nebenziel soll gezeigt werden, dass mit entsprechenden Geschwindigkeitsoptimierungen auch unter Python eine komplexe Echtzeitanimation möglich ist. Im 4. Kapitel werden dazu geeignete Maßnahmen näher betrachtet.

# 3 Die Testumgebung

Zunächst sollte die Testumgebung „Leuchtturm Demo[1]“ (lightHouseXXXX) heruntergeladen und entpackt werden. Dabei sind folgende Dateien zum Verständis dieser Einführung notwendig:

- init.py (Startdatei: aufrufen mit python init.py)
- objects3D.py
- pyWorld.py

Das Verzeichnis texture enthält die Grafiken, die später als Textur über die Objekte gelegt werden. Es sollte zum ausprobieren der Testumgebung vorhanden sein, findet ansonsten aber keine weitere Beachtung. Dateien mit der Endung .pyc sind vorübersetze Python Programme. Sie werden selbständig vom Interpreter erzeugt und werden ebenfalls nicht weiter beachtet.

Um die Testumgebung möglichst übersichtlich zu gestalten, ist sie in drei Klassen unterteilt. Die erste Klasse beinhaltet die Initialisierung von OpenGL und trägt den Namen InitGL. Grafische Objekte und objektspezifische Operationen sind in der Klasse Objects3D untergebracht und die Klasse VirtualWorld beschreibt die Welt und beinhaltet Methoden zur Interaktion.

1 http://einstein.informatik.uni-oldenburg.de/forschung/py_ogl/

## 3.1 Initialisierung

In der Initialisierungsroutine InitGL werden die Parameter gesetzt, die für die Nutzung von OpenGL wichtig sind. An dieser Stelle werden nur die verwendeten Einstellungen erklärt, die vollständige Liste ist z.B. dem Redbook [OGL97] zu entnehmen.

Mit den ersten vier GLUT Anweisungen werden die Eigenschaften des OpenGL Fensters eingestellt. Dies sind neben der Fenstergröße und -position sowie einer Fensterüberschrift, die Anweisungen GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH. GLUT_DOUBLE schaltet den doppelten Grafikpuffer an, um den Bildaufbau verdeckt zu zeichnen. GLUT_RGBA aktiviert den RGB Modus inklusive Alphakanal der Grafikkarte. Welche Eigenschaften damit genau verbunden sind hängt von der Einstellung des Betriebssystems und der verwendeten Grafikhardware ab. Mit GLUT_DEPTH wird ein Tiefenpuffer für das Fenster angefordert, um im späteren Verlauf entscheiden zu können, ob ein Objekt von einem anderen verdeckt wird und ggf. nicht gezeichnet werden muss.

Nachdem der Tiefenpuffer angefordert wurde sollte dieser zunächst gelöscht werden. Ein partielles Löschen ist möglich, indem ein Parameter kleiner eins gewählt wird. Für die Testumgebung wird mit glClearDepth(1.0) der gesamte Puffer gelöscht. Als letzte Aktion muss der Puffer noch aktiviert werden, dies geschieht mit dem Befehl glEnable und dem Parameter GL_DEPTH_TEST. Da in OpenGL zunächst alle rechenintensiven Funktionen deaktiviert sind, muss auch die Tiefenpufferüberprüfung explizit eingeschaltet werden.

Die nächsten drei Einstellungen entsprechen der Grundeinstellung, da OpenGL aber einem Zustandsautomaten entspricht, sollten wichtige Einstellungen vor dem Gebrauch bestätigt werden. In diesem Fall heißt das, die Farbe zum Löschen wird auf schwarz gesetzt, das Schattierungsmodell wird per GL_SMOOTH auf Gouroud-Shading eingestellt und damit sich alle Operationen auf die Objekte der Testwelt beziehen wird der Matrixmodus auf Modellsicht geschaltet.

Zum erhellen der Szene wird mindestens eine Lichtquelle benötigt. Jede von ihnen muss explizit mit glEnable(GL_LIGHTx) eingeschaltet werden, wobei x einen Wert zwischen 0 und 7 annehmen kann. Einige Systeme lassen auch mir als 8 Lichtquellen zu, dies ist seitens OpenGL aber nicht garantiert. Zusätzlich lassen sich die Eigenschaften jeder Quelle genauer beschreiben.

Für die Leuchtturmszene werden zwei Lichtquellen vorbereitet. Die erste soll später den Mond darstellen und wird mittels glLightfv(GL_LIGHT0, Typ, Wert) spezifiziert. Dabei soll der Mond die Szene durch etwas Umgebungslicht aufhellen. Für Typ wird entsprechend GL_AMBIENT und als Wert das Tupel (0.2, 0.2, 0.2, 1) eingetragen, was einem dunklen Grau in der RGBA[2] Schreibweise entspricht. Eine ausgeprägte diffuse Reflexion wird mit GL_DIFFUSE und (0.5, 0.5, 0.5, 1) erreicht. Die zweite Lichtquelle ist das Leuchtfeuer im Turm. Damit ein gerichteter Lichtstrahl nachgebildet werden kann, müssen zusätzlich einige Parameter mittels der GL_SPOT Befehle angegeben werden, eine detailierte Beschreibung hierzu befindet sich im Redbook[OGL97]. Jetzt fehlen nur noch die Positionsangaben der Lichter. Da es sich in beiden Fällen um animierte Lichtquellen handelt, werden diese erst in der späteren Szenenbeschreibung positioniert.

Die Auswirkungen des Lichts auf ein Objekt werden mit glColorMaterial(GL_FRONT, GL_DIFFUSE) spezifiziert. GL_FRONT bedeutet dabei, dass einfallendes Licht nur zur Beleuchtung herangezogen wird, wenn es auf die definierte Vorderseite trifft. GL_DIFFUSE bewirkt eine diffuse Reflexion.

Damit die Farbgebung und die oben eingestellte Beleuchtung zum tragen kommen, müssen beide noch mit dem glEnable Befehl aktiviert werden. Der letzte glEnable Befehl des Beispiels aktiviert die automatische Normalisierung der Oberflächennormalen. Dadurch geht zwar etwas Rechenzeit verloren, aber unerwünschte Beleuchtungseffekte werden minimiert.

Abschließend bleibt die Zuweisung der verschieden Methoden, mit der die Welt beschrieben und

2 Der Alphawert hat in diesem Beispiel keine Auswirkung, wird der Vollständigkeit halber aber mit angegeben und auf 1 gesetzt .

manipuliert werden kann. Dazu wird eine Instanz vom Typ VirtualWorld erzeugt und deren Methoden werden den entsprechenden GLUT Funktionen zugeordnet.

Das Hauptprogramm beinhaltet die Initialisierung der GLUT Bibliothek, das Instanziieren der Testumgebung und das Eintreten in die OpenGL Hauptschleife.

## 3.2 Objekte

In diesem Abschnitt werden nicht alle Methoden der Klasse im Detail erklärt. Vielmehr werden Vorgehensweisen zur Objekterzeugung anhand von Beispielen erläutert. Dabei spielen die Methoden loadTexture, calcNormal und points2Vector eine wichtige Rolle, weswegen sie im Vorfeld beschrieben werden.

### 3.2.1 Methode „loadTexture“

Mit Hilfe der Image-Bibliothek wird eine Texturdatei geladen, deren Ausmaße bestimmt und in RGB Rohdaten umgewandelt. Dabei ist zu beachten, das Höhe und Breite der Grafik einer Zweierpotenz entsprechen, z.B. 256x128 Pixel. Mit gluScaleImage wird eine Umrechnungsfunktion für Bilder beliebiger Größe bereitgestellt, diese wird hier aber nicht weiter betrachtet.

Es folgt eine Standardprozedur zur Erzeugung der Textur, die dem Redbook [OGL97] entnommen wurde. Rückgabeparameter ist eine Liste mit drei Textur Qualitätsstufen, die zur Wiederverwendung der geladenen Texturen genutzt wird.

### 3.2.2 Berechnung der Normalen

Die Methode points2Vector nimmt zwei Raumkoordinaten entgegen und liefert einen Verbindungsvektor zurück. Der erzeugte Vektor zeigt jeweils vom ersten Raumpunkt auf den zweiten. Diese Orientierung spielt bei der Berechnung der folgenden Oberflächennormalen eine Rolle.

Der Normalenvektor einer Oberfläche wird durch das Kreuzprodukt in der Methode calcNormal brechnet[3]. Zu beachten ist, dass es sich um ein rechtsdrehendes System handelt und das der zurückgegebene Vektor noch nicht normalisiert ist. Dies geschieht durch die automatische Normalisierung, die in Abschnitt 3.1 beschrieben wurde.

### 3.2.3 Quader

Die Methode cuboid veranschaulicht die Möglichkeit, mit OpenGL Grafikprimitiven komplexere Objekte zu erstellen. Dazu wird als erstes die gewünschte Primitive, in diesem Fall Rechtecke, mit glBegin(GL_QUADS) initiiert. Bis zum nächsten glEnd() werden je vier aufeinander folgende glVertex Befehle als Eckpunktkoordinaten eines Rechtecks interpretiert. Wichtige Zusatzinformationen, wie Texturkoordinaten, Normalenvektoren usw., können vor jedem Knoten eingefügt werden und gelten für alle nachfolgenden. Im Beispiel wird für jede Seite ein Normalenvektor mit glNormal definiert, damit Lichtreflexionen richtig berechnet werden können. Der Normalenvektor gilt jeweils für die vier Folgeknoten und wird dann neu gesetzt.

### 3.2.4 Der Leuchturm

Die Methode lightHouse definiert ein Objekt, welches zunächst nicht gezeichnet wird. Was es genau damit auf sich hat, wird in Abschnitt 4.1 zum Thema Displaylisten beschrieben. An dieser Stelle können zunächst die beiden ersten und letzten Zeilen im Beispielquelltext ignoriert werden.

Begonnen wird mit der Sicherung der aktuellen Arbeitsmatrix gefolgt von einer Rotation mit 270° um die x-Achse, damit der Turm aufrechtstehend modelliert werden kann. Der erste Parameter von glRotatef stellt den Winkel ein, die letzten drei geben die Koordinaten des Vektors an, um den die

3 In OpenGL werden abweichend vom Standard nicht die Normalen der Fläche sondern die der Knotenpunkte betrachtet, in diesem Beispiel ist der Unterschied aber vernachlässigbar.

Szene rotiert wird, in diesem Fall die x-Achse. Abwechselnd werden mit Hilfe der GLU Funktion gluCylinder die Segmente des Turm zusammengesetzt. Als Parameter werden dabei folgende Werte benötigt:

1. Ein mit gluNewQuadric() erzeugtes Objekt
2. Durchmesser am Fuss des Zylinders
3. Durchmesser am oberen Zylinderende
4. Höhe des Zylinders
5. Anzahl der Unterteilungen um die Höhenachse
6. Anzahl der Unterteilungen entlang der Höhenachse

Jeweils vor dem Zeichnen des Zylinders muss die gewünschte Farbe eingestellt werden, dies geschieht mit glColor. Der Turm beginnt am Fuss mit einem kräftigen Rot, dass entspricht als Parameter in der RGB Schreibweise (1,0,0), danach folgt ein weißer Zylinder usw. Da geometrische Objekte aus der GLU und GLUT Bibliothek stets am Koordinatenursprung gezeichnet werden, ist jeweils ein Vorschub entlang der z-Achse notwendig, welcher mit glTranslatef erreicht wird.

Der untere Teil der Turmplattform wird mit einer gluDisk abgeschlossen, da die oben erzeugten Zylinder an beiden Enden offen sind und man sonst von oben in den Turm hineinschauen könnte. Die Parameter, die zur Erzeugung einer Scheibe benötigt werden sind wieder das mit gluNewQuadric() erzeugtes Objekt[4], innerer und äußerer Radius sowie Anzahl der Sektoren und Ringe um die z-Achse.

Das Leuchtfeuer im Turm wird mit Hilfe eines gelben Zylinders nachgebildet. Damit es den Anschein erweckt, dass es selbstleuchtend ist, muss die Materialeigenschaft geändert werden. Um den aktuellen Wert später wieder zurücksetzen zu können und ein Leuchten aller weiteren Objekte zu verhindern, wird dieser zunächst mit glGetMaterialfv(GL_FRONT, GL_EMISSION) gesichert. Danach wird mit glMaterialfv(GL_FRONT, GL_EMISSION, (1,1,0,1)) der neue Wert eingestellt. GL_FRONT bedeutet dabei, dass nur die sichtbare Außenseite des Zylinders betroffen ist, GL_EMISSION erzeugt den gewünschten Leuchteffekt und (1,1,0,1) stellt den RBGA für gelb dar. Nach dem Zeichnen des Zylinders erfolgt das Zurücksetzen auf den alten Wert.

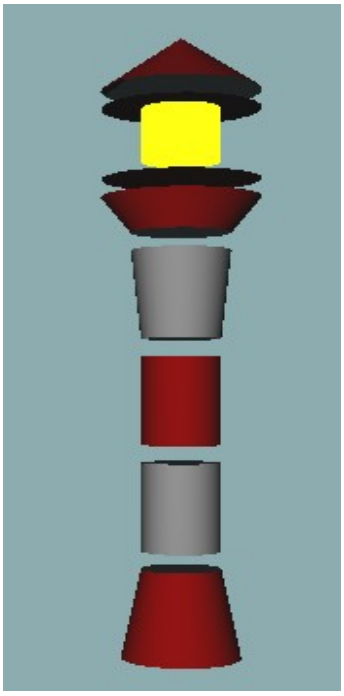
Abbildung 2 Die Einzelteile

Nach obigem Muster folgt das Dach, wobei die zweite verschließende Scheibe noch um 180° gedreht werden muss, damit die Vorderseite nach unten zeigt und dem korrekten Beleuchtungsverhalten entspricht. Zum Schluss wird mit glPopMatrix() die ursprüngliche Arbeitsmatrix wieder hergestellt. Die Auswirkungen der einzelnen Schritte sind in Abbildung 2 zu sehen, wobei der Quelltext natürlich die korrekten Abstände zwischen den Segmenten beinhaltet.

## 3.3 Die Modellierung der Welt

Die Klasse VirtualWorld ist in die Bereiche Darstellung und Interaktion aufgeteilt.

4 Eine Wiederverwendung des Objektes quad stellt in diesem Fall kein Problem dar.

### 3.3.1 Darstellung

Für die Darstellung der Szene sind die beiden Methoden drawScene und drawIdleScene zuständig. Wie der Name schon andeutet, wird drawIdleScene immer aufgerufen, wenn keine Benutzereingabe erfolgt. Damit Animationen der Szene auch fortgesetzt werden, wenn keine Interaktion stattfindet, ruft drawIdleScene die Methode drawScene auf.

Die Beschreibung der Szene findet in drawScene statt. Vor jedem Bildaufbau ist es wichtig, den Farb- und Tiefenpuffer zu löschen, damit keine Artefakte zurückbleiben. Dies geschieht mit glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT). Die aktuelle Matrix muss ebenfalls wieder auf den Ausgangswert gesetzt werden, um kumulative Matrixeffekte zu vermeiden. Hierfür stellt OpenGL den Befehl glLoadIdentity() zur Verfügung, welcher die Einheitsmatrix wieder herstellt.

Wie im Falle der Objekte in Abschnitt 3.2 werden nur einige Beispiele der Szene betrachtet, da dies für ein grundlegendes Verständnis ausreichen sollte.

Als erstes wird das „Schiff 1“ betrachtet. Mit glColor wird dem Schiffsrumpf eine Farbe zugewiesen, Schornstein und Deckaufbau sind von dem Objekt bereits vorgegeben und können an dieser Stelle nicht mehr verändert werden. Das Schiff wird mit glTranslatef(self.shipCoordX, 0, self.shipCoordZ) positioniert und durch glRotatef(self.shipRotate,0,1,0) um die y-Achse gedreht. Die Werte shipCoordX, shipCoordZ und shipRotate lassen sich über die im nächsten Abschnitt erwähnte Methode keyPressed per Tastatureingaben verändern und ermöglichen ein Umherfahren. Damit der Eindruck einer leichten Wellenbewegung entsteht, wird mit glRotatef(2*cos(self.rotation), 1, 0, 0.7) eine zeitabhängige, leichte Rotation um die x- und z-Achse eingefügt. Nachdem alle räumlichen Transformationen abgeschlossen sind, wird das vorgefertigte Schiffsobjekt mit glCallList(self.ship) an die gewünschte Stelle platziert.

Die umschließende glPushMatrix()/glPopMatrix() Operation sorgt dafür, dass alle Matrixberechnungen, die zur Positionierung benötigt wurden, wieder auf den Ausgangszustand zurückgesetzt werden.

Als zweites folgt der Leuchturm inklusive Licht. Der Leuchturm selber wird nach dem eben beschriebenen Schema platziert. Auffällig ist die Trennung zwischen Turm und Lichtstrahl. Dies liegt an einer OpenGL spezifischen Eigenart, nach der zunächst alle soliden Objekte gezeichnet werden müssen, bevor mit den transparenten, in diesem Fall der Lichtkegel, begonnen werden kann. Andernfalls kann es zu fehlerhaften Darstellungen kommen.

Der Lichtstrahl soll oben vom Leuchturm ausgehen und eine leichte Neigung nach unten besitzen. Dazu muss wieder der Koordinatenursprung mittels glTranslatef(5,24,0) verschoben werden. Eine zeitabhängige Rotation des Lichtkegels wird durch glRotatef(15*self.rotation, 0 ,1 , 0) erzeugt und die Neigung um 10° nach unten mit glRotatef(10,1,0,0). Mit glColor4f(1,1,0,0.3) wird der Lichtkegel gelb eingefärbt, wichtig ist diesmal der Alphawert, da er zusammen mit glBlendFunc(GL_SRC_ALPHA, GL_ONE) angibt, wie intensiv der Lichtstrahl zu sehen ist. Das *Blending,* also das Vermischen überlagerter Farbwerte, muss für die Operation explizit mit glEnable(GL_BLEND) eingeschaltet und nach dem zeichnen des Lichtkegels wieder abgeschaltet werden. Jetzt kann der Lichtkegel, simuliert durch einen gluCylinder, gezeichnet werden.

Damit der Lichtkegel auch einen Leuchteffekt verursacht, wird eine weitere Lichtquelle benötigt. GL_LIGHT1 wurde bereits in der InitGL Klasse initialisiert, so dass hier nur noch die Positionierung erfolgen muss. Da durch die vorangegangene Translation der Koordinatenursprung immer noch im oberen Teil des Leuchtturms ist, kann die Lichtquelle mit glLightfv(GL_LIGHT1, GL_POSITION, (0,0,0, 1)) an die richtige Stelle gesetzt werden. Der vierte Parameter im Tupel, die eins, gibt an, dass es sich um ein positionales Licht handelt, also eine Lichtquelle im gebräuchlichem Sinne. Direktionales Licht würde durch eine null an dieser Stelle erreicht werden. Letzteres ist aber hauptsächlich für die Erhellung einer kompletten Szene geeignet und ist eher mit Tageslicht vergleichbar. Abschließend bekommt die Lichtquelle noch die Richtung vorgegeben, in der sie abstrahlen soll, dies geschieht mit glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION,

(0,0,1)). Da die obigen Rotationen der Szene noch bis zum glPopMatrix() wirksam sind, reicht es den Strahl entlang der z-Achse auszurichten.

Als letzte Aktion muss ein glutSwapBuffers() ausgeführt werden, damit der verdeckt gezeichnete Grafikpuffer sichtbar gemacht wird.

### 3.3.2 Interaktion

Die Methoden activeMouse, passiveMouse, completeMouse und keyPressed sind weitestgehend selbsterklärend. Einige Besonderheiten sollen jedoch erwähnt werden.

Die Methoden activeMouse und passiveMouse beinhalten in ihren Übergabeparametern bei gedrückter bzw. gelöster Maustaste stets die aktuellen x- und y-Koordinaten der Maus. Dies lässt sich beispielsweise gut für freie Bewegungen nutzen. CompleteMouse hingegen wird nur aufgerufen, wenn sich der Status der Maustasten ändert. In diesem Moment werden Tastenstatus und Mauskoordinaten übergeben. Der erste Übergabeparameter kann die Werte GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON oder GLUT_RIGHT_BUTTON annehmen, werden mehrere Tasten „gleichzeitig“ betätigt, gilt die letzte Aktion. Der zweite Übergabeparameter enthält den Status GLUT_UP oder GLUT_DOWN, je nachdem ob die letzte Aktion das Drücken oder das Lösen einer Maustaste war. Im Beispiel wird diese Funktion dazu genutzt, die absoluten Koordinaten der Maus in relative umzurechnen.

Tastatureingaben werden von der Methode keyPressed bearbeitet. Sobald eine Taste betätigt wird, werden als Parameter der Tastenwert und die aktuellen Mauskoordinaten übergeben. Zu beachten ist, dass Tasten mit besonderer Funktion u.U. nicht abgefragt werden können.

# 4 Geschwindigkeitsoptimierung

Damit eine Animation fließend wirkt, sollte eine Szene mit mindestens 25 Bilder pro Sekunde gezeichnet werden. Das bedeutet, dass für einen einzelnen Bildaufbau maximal 40 Millisekunden zur Verfügung stehen. Müssen viele Objekte gezeichnet werden und gegebenenfalls aufwendige Rechenoperationen durchgeführt werden, ist Geschwindigkeitsoptimierung, speziell für eine Interpretersprache, ein wichtiger Faktor. Neben einer Optimierung der verwendeten Algorithmen bietet OpenGL bereits einige Hilfsmittel an.

## 4.1 Displaylisten

Mit Hilfe von Displaylisten lassen sich komplexe Objekte erzeugen und im Speicher der Grafikkarte ablegen, wobei ein doppelter Geschwindigkeitsvorteil erzielt wird. Zum einen ist der Zugriff auf den Speicher der Grafikkarte um ein vielfaches schneller als auf den Arbeitsspeicher des Rechners [Wiki01], zum anderen werden die Anweisungen zur Erzeugung des Objekts im Vorfeld übersetzt und benötigen so zur Laufzeit keinen zusätzlichen Rechenaufwand.

### 4.1.1 Verwendung von Displaylisten

Der Nachteil, dass es sich bei Displaylisten um statische Objekte handelt, die im Nachhinein nur noch mit einigen OpenGL Befehlen, z.B. glScale und glRotate, manipuliert werden können, wird durch die Geschwindigkeit und die einfache Anwendung wieder wett gemacht.

Die einfache Benutzung soll am Beispiel der Methode lightHouse verdeutlicht werden. Der erste Befehl, theTower=glGenLists(1), ermittelt einen freien Listenplatz und weist ihn dem Handle theTower zu. Eine neue Liste kann jetzt mit glNewList(theTower, GL_COMPILE) begonnen werden. Der Parameter GL_COMPILE bedeutet, dass die Liste fertig gestellt, aber an dieser Stelle noch nicht ausgegeben wird. Alternativ kann die Liste sofort mit GL_COMPILE_AND_EXECUTE, dargestellt werden, das ist für dieses Beispiel jedoch nicht geeignet.

Wie in Abschnitt 3.2.4 beschrieben, erfolgt nun der Zusammenbau des Turms, welcher zur Beendigung der Liste mit dem Befehl glEndList() abgeschlossen werden muss. Mit return theTower wird dem aufrufenden Programm der Handle für das Leuchtturmobjekt zurückgegeben und kann dort per glCallList(self.lightHouse)[5] beliebig häufig ausgegeben werden

## 4.2 Datenfelder

Datenfelder, im englischen Sprachgebrauch als *Vertex Buffer Objects* (VBO) oder *Vertex Arrays* bezeichnet, bieten die Möglichkeit, Knotenpunkte in einem Array abzulegen und mit einer Kantenliste die Topologie des Objektes zu beschreiben. Der effizientere Zugriff auf die Daten ermöglicht eine schneller Darstellung im Vergleich zum schrittweisen Aufbau durch einzelne Grundformen und kann, im Gegensatz zu Displaylisten, dynamisch verändert werden.

Leider werden von der Pythonbibliothek nicht alle OpenGL Funktionen unterstützt. Beispielsweise ist die Funktion glDrawRangeElements, die es ermöglicht, gezielt auf einzelne Bereiche in einem Feld zuzugreifen, in der Dokumentation zwar beschrieben aber zum jetzigen Zeitpunkt nicht implementiert.

## 4.3 Geschwindigkeitsvergleich

Mit einer abgewandelten Form der Testumgebung[6] ist es möglich, einen Kegel zu zeichnen, wobei als Parameter Höhe, Durchmesser sowie die Anzahl der Schritte[7] für eine Umrundung um die Höhenachse übergeben wird. Der Kegel wird in einer Schleife mit Hilfe von GL_TRIANGLE_FAN und den Funktionen Sinus und Cosinus aus der math Klasse erzeugt und kann direkt, als Datenfeld oder in Form einer Displaylist aufgerufen werden.

In der Untersuchung wird je ein Kegel mit einer der unten aufgeführten Methoden gezeichnet (siehe Abbildung 3) und dabei mit verschiedenen Schrittgrößen getestet. Die dafür benötigte Zeit wird mit Hilfe der timing Klasse ermittelt, die Testwerte sind Tabelle 1 zu entnehmen. Dabei zeigt sich, dass die Displaylist-Funktion mit Abstand am schnellsten ist und bei gleicher Objektgröße annähernd konstant bleibt. Die Datenfeld Variante erzielt im Falle vieler Flächen einen Geschwindigkeitsvorteil von ca. 40% gegenüber dem einfachen Verfahren.

Kegelhöhe: 30, Kegeldurchmesser 10

| ***Schritte*** | ***Schleife*** | ***Datenfeld*** | ***Displaylist*** |
|---:|---:|---:|---:|
| 20 | 123 | 90 | 17 |
| 200 | 966 | 570 | 17 |
| 500 | 2294 | 1352 | 17 |
| 1000 | 4513 | 2693 | 17 |

Tabelle 1 Geschwingkeitsvergleich in Microsekunden

Als Grundlage für die Messung diente folgendes System:

- Betriebssystem: Suse Linux 9.2
- Sprache: Python 2.3
- Prozessor: Pentium 4, 2.66GHz
- Hauptspeicher: 1GB
- Garfikkarte: GeForce4 MX/64MB

5 Voraussetzung für dieses Beispiel ist natürlich eine Instanziierung des Objektes lightHouse wie im Constructor der Klasse VirtualWorld geschehen.

6 http://einstein.informatik.uni-oldenburg.de/forschung/py_ogl/archiv/cones_bench.zip

7 Entspricht Anzahl der zu zeichnenden Flächen

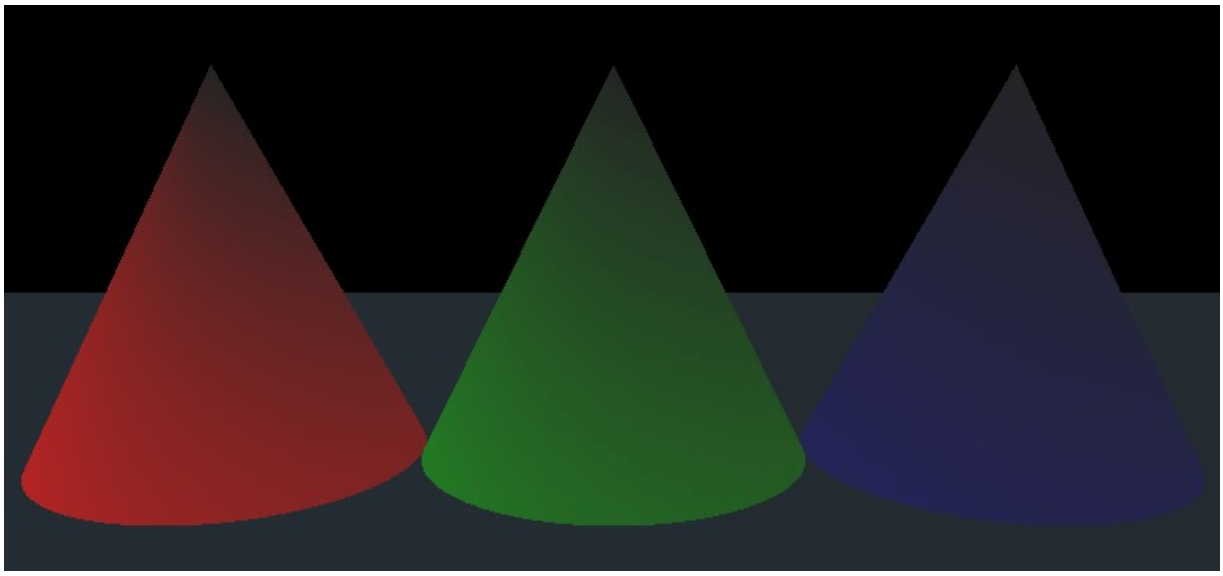

Abbildung 3 Geschwindigkeitstest

# 5 Fazit

Es hat sich gezeigt, dass sich mit Hilfe von OpenGL unter Python komplexe Szenen modellieren lassen, ohne dass ein ruckeliger Eindruck entsteht. Die Kombination OpenGL/Python bietet für Einsteiger eine schnelle und einfache Möglichkeit, dreiminensionale Animationen zu erstellen und somit Erfolge zu erzielen, die sich gerade am Anfang positiv auf die Motivation auswirken können, sich weiterhin mit der Materie zu beschäftigen.

Aufgrund der Untersuchung im 4. Kapitel liegt es nahe, dass dies nicht unbedingt für fortgeschrittene Szenen zutrifft, die nicht allein mit statischen Elementen modelliert werden können. Allerdings kann auch der erfahrene Programmierer Nutzen aus obiger Kombination ziehen. Einzelne Elemente, ansonsten für eine Interpretersprache zu komplexer Szenen, lassen sich effizient durchtesten. Ist das gewünschte Ziel erreicht, lässt sich der Code, begünstigt durch die einfache Syntax von Python, leicht in eine andere Programmiersprache portieren.

# 6 Anhang

## 6.1 Installationsvoraussetzungen

Für die Entwicklung der Testumgebung wurden neben der hardwareseiteigen OpenGL Unterstützung folgende Pakete verwendet. Das Entwicklerpaket „python-devel“ wird zum ausprobieren vermutlich nicht benötigt, ist aber der Vollständigkeit halber mit aufgeführt.

- python-devel-2.3
- python-pygame
- python-opengl
- python-numeric
- python-imaging
- python-2.3

Zu beachten ist, dass das Paket „python-opengl“ unter Suse 9.2 und 9.3 nicht vollständig ist und Befehle, beispielsweise „glutKeyboardUpFunc“ nicht implementiert sind. Dies ist speziell bei eigenen Entwicklungen zu beachten. In diesem Fall empfiehlt es sich, z.B. das Paket der Debian Distribution oder das Original von „http://pyopengl.sourceforge.net“ einzuspielen.

Die Funktion der Testumgebung unter Windows wurde seitens dritter bestätigt, leider kann hierfür keine Voraussetzungsliste angegeben werden.

## 6.2 Bedienung der Leuchtturmwelt

Tastaturbelegung:

- v : Schaltet zwischen globaler und Bootsicht hin und her
- w, s : Beiges Boot vor und zurück bewegen
- a,d : Beiges Boot nach links bzw. rechts drehen
- f : Vollbildmodus ein/aus
- Esc : Beendet das Programm

Mausbelegung: (Gilt nur für die globale Sicht!)

- Rechte Maustaste gedrückt : Bewegung entlang der z-Achse möglich
- Linke Maustaste gedrückt : Rotation um die x- und y-Achse möglich

# Literaturverzeichnis

[python] Python, http://www.python.org/
[eric3] Eric3, http://www.die-offenbachs.de/detlev/eric3.html
[OGL97] OpenGL Programming Guide, J. Neider, T. Davis, Addison-Wesley,1997
[Wiki01] Displaylisten, http://wiki.delphigl.com/index.php/Displaylisten