Übung Nr. 5:

Modelle, die mit OpenGL dargestellt werden, sollen unter Verwendung des Akkumulationspuffers Antialiasing bekommen (Abb. 1). Der größte Teil des dazu benötigten Quellcodes ist im Programm `ObjElabGLaaTx.c` enthalten; das dazugehörige MS-VC-Projekt kann unter http://homepages.thm.de/christ/ heruntergeladen werden. Als Anleitung zur Vervollständigung dient die u.a. Beschreibung der geforderten Funktionalität. Es fehlen insgesamt ca. 10 Zeilen Code in der Funktion `drawAAObj()`; diese dient als eine Art Ergänzung für die Funktion `drawObj()` und wird an ihrer Stelle von der Callback-Funktion zur Bildausgabe, `display()`, aufgerufen. In ihr finden die Berechnungen zu den Positionen und Überblendungen vor und nach dem Aufruf von `drawObj()` statt. Sonst ist sowohl das Programm als auch das dazugehörige Menü für alle Änderungen vorbereitet. (Bitte beachten: Eine Verifikation dieser und der folgenden Beschreibung anhand des verfügbaren Quellcode-Fragments und durch Beschäftigung mit dem lauffähigen Programm trägt wesentlich zum Verständnis des gesamten Zusammenhangs bei!)

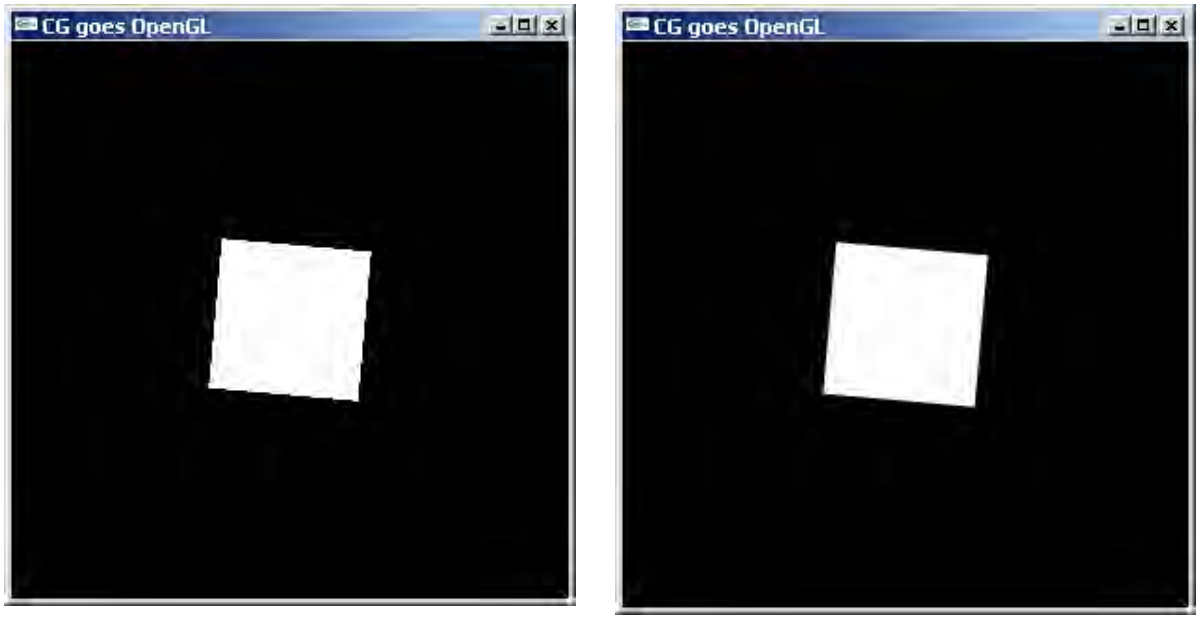

Abb. 1 Objekt ohne (li.) und mit (re.) Antialiasing

Das Programm ist ähnlich strukturiert wie die in der Vorlesung angesprochenen `accanti.c` und `accpersp.c`, die beispielhaften Implementierungen von Antialiasing, die von den Entwicklern von OpenGL zur Verfügung gestellt werden. Anitaliasing besteht auch hier in der mehrfachen Abbildung derselben Szene, jeweils mit einem Versatz von weniger als einem Pixel gegenüber ihrem exakt berechneten Abbild (Jittering, „Zittern“). So erfolgt eine Überlagerung mehrerer Abbildungen desselben Objekts mit jeweils unterschiedlicher Rasterisierung (vergleichbar einer Mehrfach-Belichtung bzw. einem verwackelten Bild). Typische, von den Entwicklern empirisch als vorteilhaft erkannte Kombinationen von Positionen (dargestellt durch die Koordinaten-Paare ihrer Verschiebungen in der Projektionsebene) finden sich in der Datei `jitter.h`, zusammengestellt als indizierte Struktur-Variablen `j2[]`, `j3[]`, `j4[]`, `j8[]`, `j15[]`, `j24[]`, `j66[]` (im folgenden allgemein: `j*[]`), mit je zwei Elementen, x und y.

Das hier zu vervollständigende Programm soll, anders als die vorgenannten, eine freie Wahl zwischen den o.a. Struktur-Variablen (d.h. von 2- bis 66fachem Multisampling) ermöglichen; zur Steuerung des Wechsels zwischen den Koordinaten-Sätzen `j*[]` sind im Menü die Tasten '+' bzw. '–' vorgesehen. Im Quellcode ist die indizierte Variable `p2jit[]` (in der Funktion `drawAAObj()`) dafür reserviert; mit jedem Index soll einer der Koordinaten-Sätze assoziiert werden. Die indizierte Variable `aaJit[]` ist bereits mit der jeweiligen Anzahl von Elementen der o.a. Struktur-Variablen `j*[]` initialisiert. Sie ist global, weil sie auch in der Funktion `key()` fürs Menü gebraucht wird.

Darüber hinaus soll der Versatz zwischen den einzelnen (2 bis 66) Teilbildern im geglätteten Ergebnisbild beliebig variabel sein, steuerbar über die Tasten '>' bzw. '<'.

Neben diesem (ansatzweise realisierten) Antialiasing durch Versetzen der Szene bzw. der Objekte soll auch eine Alternative implementiert werden, bei der die Projektionsfläche bzw. der Viewport (weiterhin gegenüber dem Augenpunkt) verschoben wird. Daraus ergeben sich zwei Teilaufgaben.

1 Antialiasing durch Versetzen der Szene gegenüber dem Augenpunkt

In der Flächenmodell-Darstellung der anfänglichen (unvollständigen) Version liefert die Zuschaltung von Antialiasing keine Veränderung am Ergebnisbild. Ein Grund hierfür hat zuvor eine Compiler-Warnung erzeugt: Die indizierte Variable `p2jit[]` wird weder mit Werten besetzt noch verwendet. Hier sollte auch der erste Schritt zur Lösung erfolgen, indem `p2jit[7]` mit den Zeigern auf die indizierten Struktur-Variablen `j*[]` initialisiert wird. Das kann in der Deklarationszeile geschehen (vgl. `aaJit[]`-Deklaration/ Definition).

Diese erste Korrektur beseitigt zwar die o.a. Compiler-Warnung; sie ändert aber nichts an der Grafik; denn die mit `p2jit` verbundenen Subpixel-Positionen gehen nicht in die Berechnungen in `drawAAObj()` ein. Maßgeblich für die Versetzung eines Objektes innerhalb des Multisampling sind dort die Variablen `dx` und `dy`, wie man unschwer anhand des dazugehörigen Aufrufs `glTranslatef()` erkennt. In diesem Aufruf repräsentieren die Variablen `dx` und `dy` Verschiebungen in Weltkoordinaten (in dem sog. Window, streng genommen: in der Projektionsebene). Eine Verschiebung um `1.0` (das ist der max. Abstand zwischen zwei Positionen in der Mehrfach-Abbildung) soll dabei einen Versatz von einer Pixelbreite bzw. -höhe im digitalisierten Fenster (Viewport) bedeuten. `dx` und `dy` sollen deshalb ein Produkt der vorgeschlagenen Verschiebungen (`j*[].x` bzw. `j*[].y`, die alle zwischen `–.5` und `+.5` liegen) mit der (x-/y-)Kantenlänge eines Pixels ausgedrückt in Weltkoordinaten enthalten.

Zur Umrechnung der Pixelbreite bzw. -höhe in Weltkoordinaten dienen die Angaben bei der Einrichtung einerseits des Fensters (Anzahl Pixel in x- und y-Richtung) und andererseits des Sichtvolumens (Abmessungen in Weltkoordinaten der Grundfläche eines Sicht-Kegelstumpfs bei perspektivischer bzw. eines Sicht-Quaders bei orthographischer Projektion).

Ein weiterer Faktor, der in `dx` und `dy` ebenfalls eingehen soll, repräsentiert eine Gesamtskalierung (im Programm als Offset bezeichnet), um die der Versatz zwischen den Teilbildern vergrößert oder verkleinert dargestellt werden soll. Dafür ist die globale Variable `aaOff` vorgesehen; sie ist mit dem neutralen Element, dem Wert `1.0` initialisiert worden. Durch Erhöhung bzw. Verringerung des Wertes von `aaOff` (über die Tasten '>' bzw. '<') sollen sich im gleichen Maß `dx` und `dy` verändern. Damit sollen schließlich die Teilbilder, die zum geglätteten Ergebnisbild beitragen, auseinander rücken. Abb. 2 zeigt ein Beispiel.

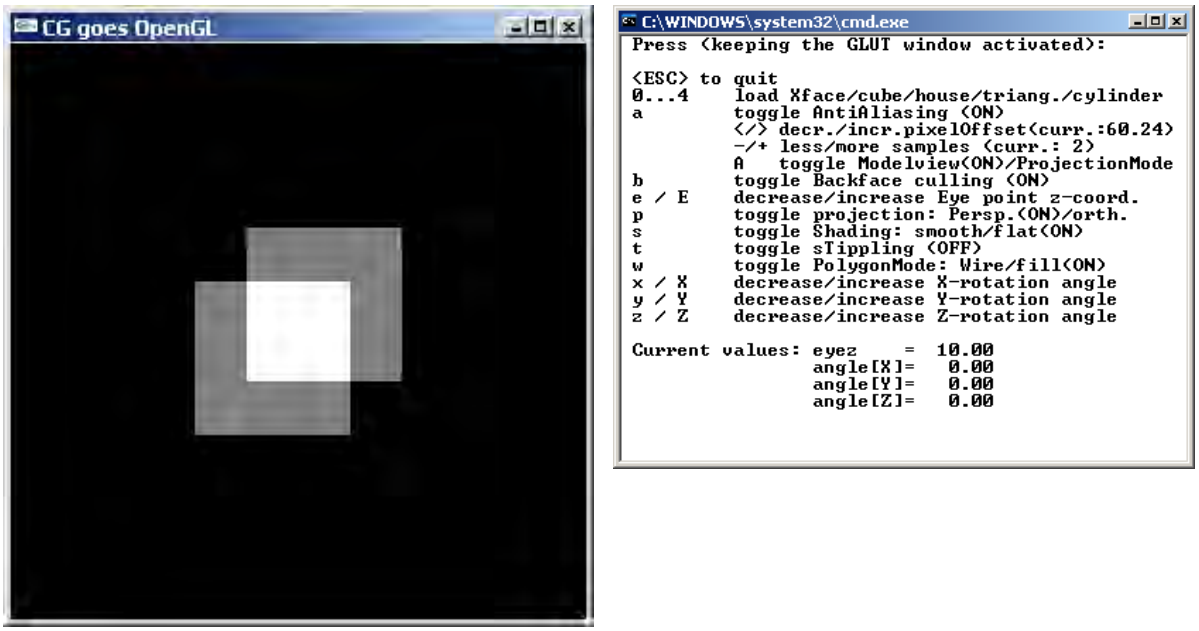

Abb. 2 Antialiasing mit Zweifach-Multisampling bei ca. sechzigfach vergrößertem Abstand zwischen den Teilbildern

Die bisher durchgeführten Verbesserungen am Programm führen jedoch zu Darstellungen nicht nach Abb. 2, sondern nach Abb. 3.

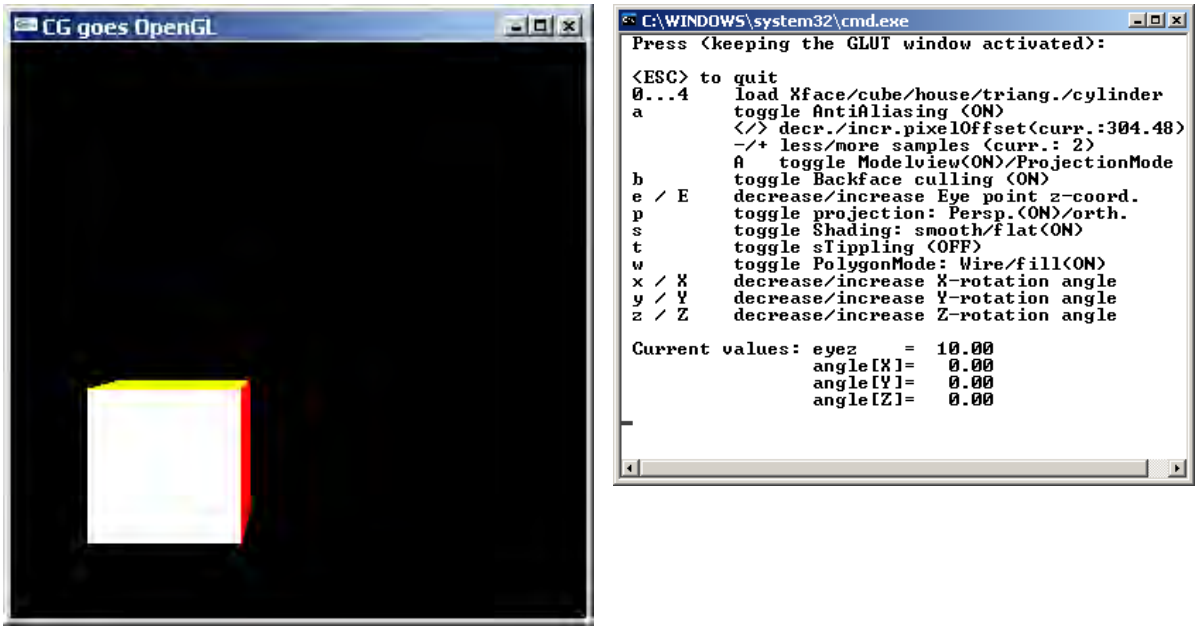

Abb. 3 Antialiasing-Teilbild bei Zweifach-Multisampling und ca. 300fach vergrößertem Abstand zwischen den Teilbildern

Ursache dafür ist wiederum, daß die einzeln Teilbilder zwar erzeugt und in den Akkumulationspuffer übertragen werden; letzterer wird aber nicht in den Farbpuffer zurückgeschrieben. Somit ist nur das jeweils zuletzt berechnete Teilbild zu sehen. Es fehlt der hierzu benötigte OpenGL-Aufruf.

Nach den besprochenen Code-Änderungen sind Darstellungen wie Abbn. 1 und 2 möglich.

2 Antialiasing durch Versetzen des Viewport gegenüber dem Augenpunkt

Die in der ersten Teilaufgabe implementierte Technik zeigt in der Regel den gewünschten Effekt. Eine Schwäche des Verfahrens besteht jedoch darin, daß die einander überlagerten Teilbilder nicht identisch sind, wie die Darstellung in Abb. 3 verdeutlicht. Genau betrachtet handelt es sich nicht um die mehrfache Überlagerung der Ansicht einer Szene, sondern um die Überlagerung mehrerer Ansichten einer Szene. Dies kann in Extremfällen, vor allem bei schmalen Objekten („Rasierklinge“) mit starken Kontrasten in ungünstiger Lage und kleiner Entfernung, zu irritierendem Farbschimmer entlang der Objektkanten führen.

Dieser Mangel soll nun behoben werden, indem die Funktion `drawAAObj()` um einen zusätzlichen Antialiasing-Modus erweitert wird, bei dem der Viewport statt der Szene gegenüber dem Augenpunkt verschoben wird. Im Menü (Funktion `key()`) ist zu erkennen, daß hierfür die globale Variable `aaMod` bereits für eine Umschaltung zwischen den beiden Modi berücksichtigt wird.

Da es hier wieder um pixelweise Verschiebung der Darstellungen geht, können zur Erweiterung von `drawAAObj()`die im 1. Teil dieser Übung ermittelten Größen `dx` und `dy` in adäquater Weise (Vorzeichen!) eingesetzt werden; durch Umschalten zwischen den beiden Modi bei großen Abständen läßt sich dann die fast pixelgenaue Übereinstimmung zwischen den beiden Verfahren vorführen (vgl. Funktionsmuster). Es sei daran erinnert, daß Positionierungen und Rückführungen in eine anfängliche Position (d.h.: in jene vor Beginn aller Transformationen) in OpenGL mit den gleichen Funktionsaufrufen vorgenommen werden. Für die Unterscheidung, ob eine Transformation der Szene (bzw. des Objektes) oder des Augenpunktes gewünscht wird, sorgt die Einstellung des „Matrixmodus“, der (einheitlich von Entwicklern) gewohnheitsmäßig auf Modeltransformation (sog. Modelsicht) eingestellt wird.

Empfehlungen für die weitere Auseinandersetzung mit dem Programm

Es ist ratsam, die Entwicklung (wie voreingestellt) im Darstellungsmodus mit Einzelpuffer (Single Buffer) durchzuführen. Da hier das Ergebnisbild im ständig angezeigten Puffer entsteht, kann beim Debuggen an beliebiger Stelle des Programms durch Einfügen der Anweisung `glFinish()` der Stand der Grafik-Entstehung besichtigt werden. Nach abgeschlossener Entwicklungsarbeit lohnt es sich auch, die Wirkung des Double Buffering (gerade bei Multisampling mit vielen Teilbildern) auszutesten.

Die Komplexität des Verfahrens eignet sich gut, Gedanken über die Programm-Struktur anzustellen: Fragen der Aufteilung zwischen globalen oder lokalen Variablen, Nutzung gemeinsamer Anweisungen für beide Antialiasing-Modi und -vor allem- die Reihenfolge, in der das Programm die benötigten Transformationen in der Grafik-Pipeline ausführt, können gut beobachtet und diskutiert werden, ebenso wie der Verlauf der notwendigen Schleifen für die logisch nacheinander erfolgenden Schritte:

- Farbpuffer-Löschung für jedes Teilbild
- Objekt-Animation und -Positionierung
- Objekt-Versetzung (bei Antialiasing durch Versetzen der Szene)
- Perspektivische o. orthogonale Projektion (ggf. mit Viewport-Versetzung für Antialiasing)
- Objekt-Darstellung über seine Ecken, Flächen, Farben etc.
- Teilbild-Übertragung in den Akkumulationspuffer.
- Ergebnisbild-Übertragung aus dem Akkumulationspuffer in den Farbpuffer.