



Programmieren mit *VTK*

Manfred Brill

Oktober 2005

Inhaltsverzeichnis

1	Die Architektur der VTK	1
2	Eine Beispiel-Pipeline	2
3	VTK-Anwendungen erstellen	4
3.1	VTK und Microsoft Visual C++	4
3.2	VTK mit <i>make</i> und <i>g++</i>	5
4	Quellcode-Installation	8
4.1	CMake	8
4.2	VTK mit <i>Microsoft Visual C++ 6.0</i>	9
4.3	VTK mit <i>g++</i> auf <i>cygwin</i> oder <i>LINUX</i>	10
4.4	Eigene VTK-Projekte	11
	Literatur	12

Zusammenfassung

VTK ist als Begleitsoftware zur Monographie [SML97] entstanden. Inzwischen steht damit eine mächtige Sammlung von Algorithmen und Datenstrukturen für die Visualisierung zur Verfügung. Dieses Dokument soll Ihnen helfen, erste kleine Visualisierungspipelines zu erstellen und mit Hilfe von *Microsoft Visual C++*, *g++* und *cygwin* oder *LINUX* zum Laufen zu bringen. Dabei gehen wir zu Beginn davon aus, dass *VTK* bereits installiert ist. Anschließend, im zweiten Teil des Dokuments, betrachten wir einige Details der Installation von *VTK* auf *Microsoft Windows* oder *LINUX*.

Kapitel 1

Die Architektur der VTK

VTK können wir grob in zwei große Abschnitte teilen. Auf der einen Seite steht das *graphics model*. Dies stellt eine abstrakte Realisierung der Computergrafik-Pipeline dar; es stehen Funktionen vergleichbar der *GLUT* zu Verfügung, um ein Grafikfenster zu öffnen, es gibt Renderer und insbesondere grafische Objekte mit Attributen.

Daneben, schließlich ist das „V“ ganz groß geschrieben bei VTK gibt es die Visualisierungspipeline. Es gibt Datenobjekte, die von der virtuellen Klasse `vtkDataObject` abgeleitet sind. Hiermit stehen eine Vielzahl von Datenstrukturen für die Visualisierung zur Verfügung – von Containern für Punkte bis hin zu unstrukturierten räumlichen Gittern mit skalaren, vektoriellen und tensoriellen Attributen. Die Pipeline wird durch Filter, Klassen, die die Daten verarbeiten realisiert. Diese Klassen sind von `vtkProcessObject` abgeleitet. Mehr dazu finden wir in [SML97, BB05].

Die Datenstrukturen der VTK treten häufig nicht explizit im Quelltext auf. Die Filterklassen haben alle die Funktionen `::SetInput()` und `::GetOutput()`. Damit kann die Pipeline „zusammengesteckt“ werden. In einer Pipeline finden Sie sehr häufig Aufrufe der Form

```
filter2->SetInput(filter1->GetOutput());
```

Grafisch entspricht dies dem Datenfluss-Diagramm in Abbildung 1.1. Die Klasse `filter1` hat als Ausgabe eine VTK-Datenstruktur, die von `vtkDataObject` abgeleitet ist. Diese Klasse wird von `filter2->SetInput()` als Übergabeparameter akzeptiert.

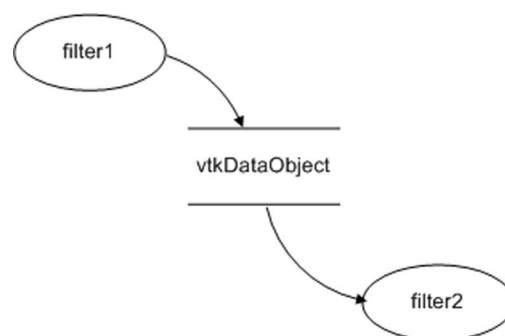


Abbildung 1.1: Datenfluss-Diagramm für die Konstruktion einer Visualisierungspipeline in VTK

Mit Hilfe von Ableitungshierarchien und der strengen Typung während der Kompilierung kann gut überprüft werden, ob die Pipeline korrekt konstruiert wurde.

Kapitel 2

Eine Beispiel-Pipeline

Häufig lesen wir in einer Visualisierungspipeline die Daten, die grafisch dargestellt werden sollen aus einer Datei ein. Dazu enthält VTK eine ganze Reihe von Filtern, die die verschiedensten Dateiformate einliest und als Ergebnis eine entsprechende Datenstruktur hat. Der zentrale Punkt für den Übergang von der Visualisierungs-Pipeline und der grafischen Ausgabe stellt der *Mapper* dar. Dies sind in VTK Klassen, die von `vtkDataSetMapper` abgeleitet werden. Je nach Pipeline verwenden Sie entsprechende Klassen, die die Visualisierungsdaten auf grafische Objekte abbildet werden. Sehr häufig werden sie dabei die Klasse `vtkPolyDataMapper` verwenden.

Betrachten wir ein Beispiel. Wir wollen ein Ebenenstück instanzieren, das mit einer Textur dargestellt werden soll.

```
// Bild lesen und einer Instanz von vtkTexture übergeben
vtkBMPReader *bmpReader = vtkBMPReader::New();
  bmpReader->SetFileName("spielbrett.bmp");
vtkTexture *texture = vtkTexture::New();
  texture->SetInput(bmpReader->GetOutput());
  texture->InterpolateOn();

// Ein Ebenenstück instanzieren.
vtkPlaneSource *plane = vtkPlaneSource::New();
  plane->SetOrigin(0.0,0.0,0.0);
  plane->SetPoint1(10.0,0.0,0.0);
  plane->SetPoint2(0.0, 10.0, 0.0);
```

Mit Hilfe einer Instanz von `vtkBMPReader` wird ein Bitmap eingelesen und an eine Instanz von `vtkTexture` übergeben. Dann erzeugen wir mit `vtkPlanesource` ein Ebenenstück. Dabei wird die „linke untere Ecke“ und die beiden Richtungsvektoren angegeben. Diese Klasse erzeugt auch geeignete Texturkoordinaten.

```
vtkPolyDataMapper *planeMapper = vtkPolyDataMapper::New();
  planeMapper->SetInput(plane->GetOutput());
vtkActor *planeActor = vtkActor::New();
  planeActor->SetMapper(planeMapper);
  planeActor->SetTexture(texture);
```

Das Ebenenstück hat eine konkrete geometrische Bedeutung, so dass im nächsten Schritt bereits mit Hilfe einer Instanz von `vtkPolyDataMapper` die Abbildung auf ein grafisches Objekt durchgeführt werden kann.

Grafische Objekte in VTK sind von der Klasse `vtkProp` abgeleitet. Ein dreidimensionales Objekt, das aus einem polygonalen Netz besteht ist durch `vtkActor` realisiert. Dieser Instanz wird die Ausgabe von `planeMapper` und die Textur übergeben. Möglich sind hier auch Angaben über weitere grafische Attribute wie `ambiente` und `diffuse` Farbe oder Angaben für die Anwendung

interpolativer Schattierungsverfahren.

```
// Fenster, Renderer ...
vtkRenderer *ren = vtkRenderer::New();
ren->AddActor(planeActor);
ren->SetBackground(0.9,0.9,0.9);
ren->GetActiveCamera()->Elevation(-30);
ren->GetActiveCamera()->Roll(-20);
ren->ResetCameraClippingRange();

vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->AddRenderer(ren);
renWin->SetSize(800,800);
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);

renWin->Render();
iren->Start();
```

Jetzt fehlt noch ein Fenster für die Ausgabe, ein Renderer, die Sicht und eine Instanz der Klasse `vtkRenderWindowInteractor`, in der Interaktionen mit der Anwendung definiert sind, wie der Tastatur-Shortcut „q“ für das Beenden des Programms. Es ist auch möglich, Pipelines zu konstruieren, die als grafische Ausgabe direkt eine *VRML*- oder *RIB*-Datei oder eine Bitmap produzieren. Als Datenfluss-Diagramm ist ein Teil der Pipeline bis zur Instanz von `vtkActor` in Abbildung 2.1 dargestellt.

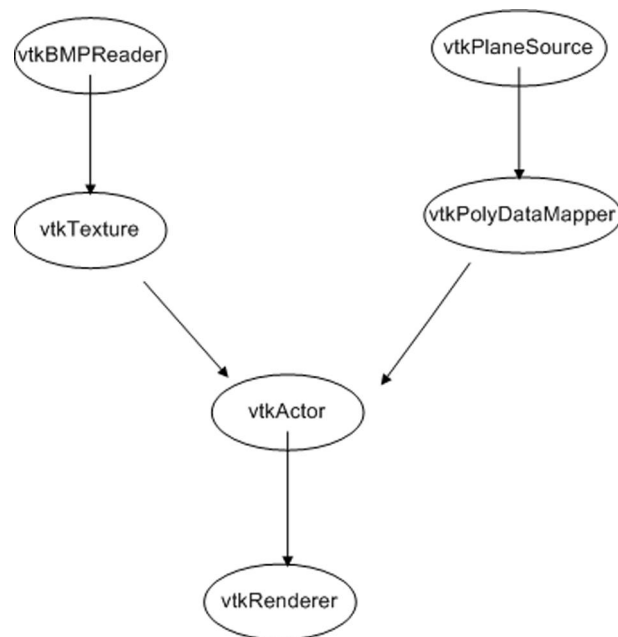


Abbildung 2.1: Datenfluss-Diagramm der Beispiel-Pipeline.

Kapitel 3

VTK-Anwendungen erstellen

3.1 VTK und Microsoft Visual C++

So lange wir nicht die in *VTK* enthaltenen patentierten Algorithmen wie *Marching Cubes* benötigen können wir mit Microsoft Windows auf www.vtk.org zur Verfügung stehenden Binärdateien zurückgreifen. *VTK* bietet neben der *C++*-Bindung auch Sprachbindungen an *Java*, *tcl* und *Python* an. Sie müssen sich bei der Installation entscheiden, welche Bindungen Sie installieren möchten. Die Installationsdateien sind zur Zeit, für die Version 4.2, in fünf Teile aufgeteilt. Für die Binär-Installation benötigen Sie auf jeden Fall `vtk42Core` und `vtk42Cpp`. *VTK* enthält eine Vielzahl von Beispielen. Die dafür benötigten Eingabedaten können Sie als `vtkData` ebenfalls installieren.

Typischerweise werden diese Dateien unter dem Pfad `C:\Programme\vtk` installiert. Wichtig für das Übersetzen und Binden von *C++*-Programmen mit *VTK* sind die Unterverzeichnisse `include\vtk` und `lib\vtk`. Dort finden Sie die Header-Dateien und die Bibliotheken zum Binden. *VTK* besteht aus einer ganzen Reihe von Bibliotheken, die wir in der richtigen Reihenfolge angeben müssen, falls statisch gebunden wird.

Es ist möglich, eine Windows-Applikation mit *VTK* zu implementieren; mehr Einzelheiten dazu finden Sie in [SMAL00]. Wie im Fall von *OpenGL* kann auch eine Konsolen-Applikation implementiert werden. Diese Wahl hat den Vorteil, dass der Quelltext identisch ist, falls wir uns entscheiden, die Pipeline unter *LINUX* oder *cygwin* und *g++* zu compilieren.

Damit Visual C++ die Header-Dateien und die Bibliotheken findet, müssen wir das Verzeichnis `include\vtk` in den Suchpfad von Visual aufnehmen. Dazu wählen Sie `Extras | Optionen` und im dann ausgegebenen Fenster den Reiter *Verzeichnisse*. In den Abbildungen 3.1 und 3.2 sehen Sie eine Darstellung dieses Fensters.

Diese Einstellungen werden in Ihrem Profil gespeichert; solange Sie die Verzeichnisstruktur von *VTK* nicht verändern oder eine neue Version installieren, müssen Sie diese Einstellungen nicht mehr ändern.

Nehmen wir an, Sie haben den Quelltext für unser kleines Beispiel mit einer Textur und einem Ebenenstück in Microsoft Visual C++ in eine Konsolenapplikation eingefügt. Dann können Sie bereits übersetzen; und Sie sollten auch keine Fehlermeldung erhalten. Beim Versuch zu Binden erhalten Sie allerdings viele *unresolved externals*. Sie haben dem Compiler zwar mitgeteilt wo sich die Bibliotheken befinden, allerdings nicht, wie sie heißen. Dies müssen Sie für jedes Projekt durchführen. Dafür wählen Sie das Menü `Projekt | Einstellungen` und wählen *Alle Konfigurationen*. Im Reiter `Linker` müssen Sie wie in Abbildung 3.3 die folgenden Bibliotheken im Feld `Objekt-/Bibliothek-Module` eingeben:

```
vtkRendering.lib vtkGraphics.lib vtkImaging.lib vtkIO.lib vtkFiltering.lib vtkCommon.lib
```

Jetzt können Sie erfolgreich binden und das Programm ausführen.

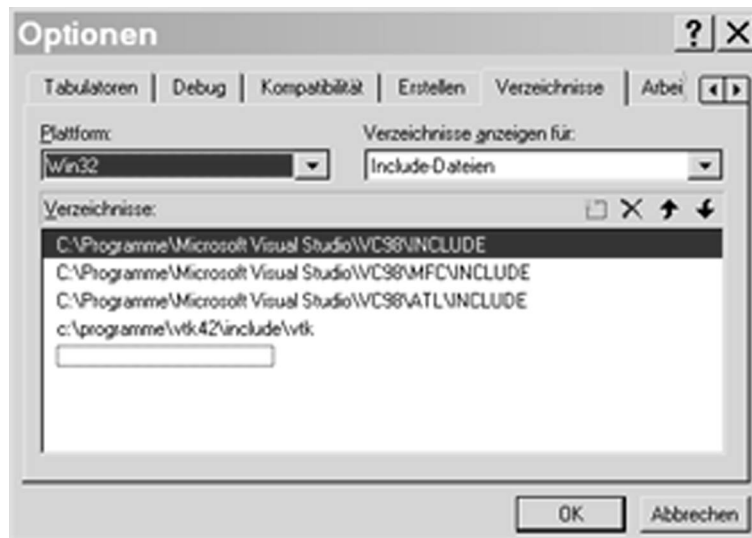


Abbildung 3.1: Aufnahme des Verzeichnisses für Header-Dateien in Microsoft Visual C++

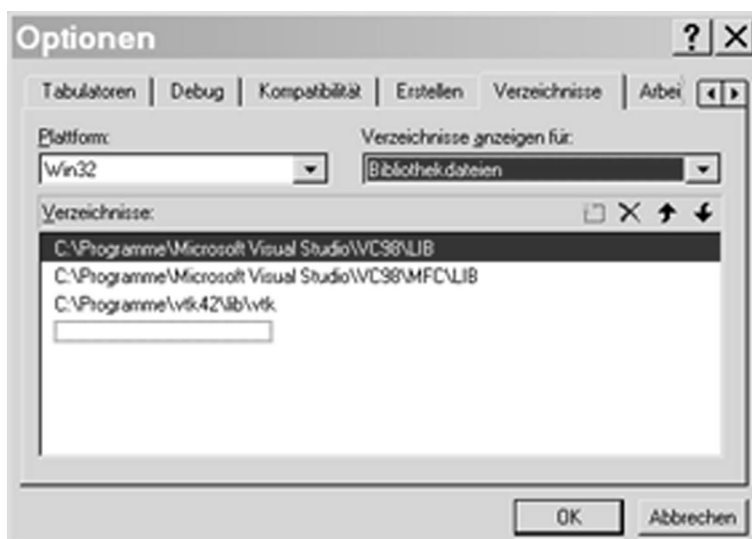


Abbildung 3.2: Aufnahme des Verzeichnisses für die Bibliotheken in Microsoft Visual C++

3.2 VTK mit *make* und *g++*

Wollen Sie mit der VTK mit dem GNU-Compiler arbeiten, dann empfiehlt sich die Arbeit mit einem Makefile. Mehr zu Makefiles finden Sie in [Bri03b]. Angenommen, wir wollen die kleine Pipeline aus dem Einführungsbeispiel übersetzen und binden, dann können wir dafür den folgenden Makefile verwenden:

```
CXX = g++
#
DEBUG      = -g
OPTIMIZE   =
VTKHEADER = -I/usr/local/include/vtk
CXXFLAGS  = -I. ${VTKHEADER} ${DEBUG} ${OPTIMIZE}

VTK_LIBDIR = -L/usr/local/lib/vtk
VTKLIBS    = -lvtkRendering -lvtkGraphics -lvtkImaging\
```

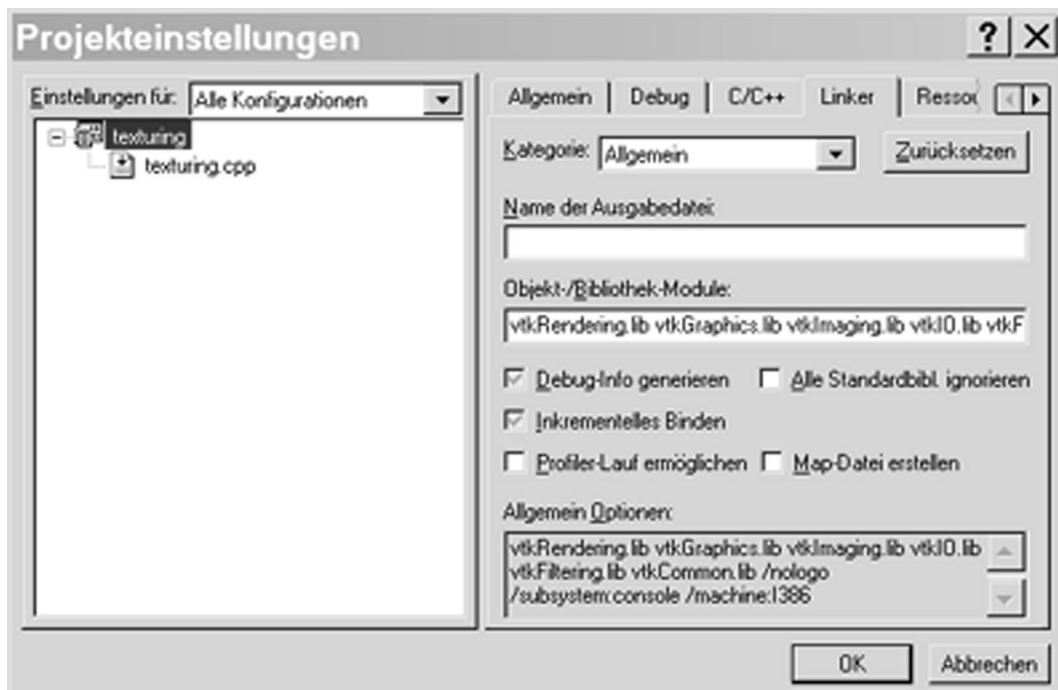


Abbildung 3.3: Angabe der für das Binden benötigten VTK-Bibliotheken in Microsoft Visual C++

```

        -lvtkIO -lvtkFiltering -lvtkCommon
DL_LIBS    = -lopengl32 -lgdi32

all : pipeline

pipeline.o : pipeline.cpp
    ${CXX} -c ${CXXFLAGS} $<

pipeline : pipeline.o
    ${CXX} -o $@ ${CXXFLAGS} $< ${VTK_LIBDIR} ${VTKLIBS} ${DL_LIBS} -lm

```

Dabei gehen wir davon aus, dass der Quelltext im gleichen Verzeichnis wie dieser Makefile steht und in der Datei `pipeline.cpp` abgespeichert wird. Das ausführbare Programm trägt dann den Namen `pipeline`. Der Makefile setzt voraus, dass Sie *VTK* in `/usr/local` installiert haben. Dann stehen die Header-Dateien in `/usr/local/include/vtk` und die Bibliotheken in `/usr/local/lib/vtk`. Sie müssen die Bibliotheken

`libvtkRendering.a libvtkGraphics.a libvtkImaging.a libvtkIO.a libvtkFiltering.a libvtkCommon.a` anbinden.

Tip:

Der dargestellte Makefile enthält bereits ein Make-Target `all`. Wenn Sie auch noch das Target `clean` hinzufügen, dann können Sie damit ein *Eclipse*-Projekt erstellen.

Auf der Website www.vtk.org gibt es für die Installation von *VTK* auf *LINUX* oder *cygwin* nur die Möglichkeit, die Quellen selbst zu übersetzen. Dazu benötigen Sie außer dem Archiv mit den Quellen auch das Freeware-Werkzeug *CMake*. Mehr zum Installieren der *VTK* auf Quellenbasis finden Sie weiter unten in diesem Dokument.

Bei der Konstruktion von Visualisierungs-Pipelines mit der *VTK* ist es sehr hilfreich, die auf www.vtk.org angebotene Dokumentation zu installieren. Diese Dokumentation wird mit Hilfe von *doxygen* ([Bri03a]) erzeugt. Es stehen eine *HTML*-Version und eine *chm*-Version zur Verfügung.

Kapitel 4

Quellcode-Installation

4.1 CMake

Wollen Sie *VTK* selbst übersetzen, dann benötigen Sie neben den Quell-Dateien das Freeware-Paket *Cmake*. Wenn Sie bereits Erfahrung mit der Installation von Freeware-Paketen auf Quellenbasis besitzen, dann kennen Sie sicher das GNU-Programm *configure*. *CMake* ist ein Werkzeug, das unabhängig von der Zielumgebung in der Lage ist, entweder einen Makefile oder einen Arbeitsbereich für *Microsoft Visual C++* zu erzeugen. Sie erhalten *Cmake* entweder als Binär-Installation für *Microsoft Windows* oder als Quell-Installation für *LINUX* oder *cygwin* auf der URL www.cmake.org. In *Windows* gibt es dann eine interaktive Version; bei einer installierten *curses*-Version steht diese für *cygwin* und *LINUX* ebenfalls zur Verfügung. In einer Shell ist mit *cmake -i* auch eine terminalbasierte Version verfügbar.

Die Quelldistribution von *VTK* enthält Eingabedateien für *Cmake*, eine Datei *CMakelists.txt*. Dort sind sogenannte *Cache-Values* eingetragen. Abbildung 4.1 zeigt *CMake* nachdem das Verzeichnis mit den Quellen angegeben wurde. Das Zielverzeichnis für die zu erstellenden Binärdateien muss ebenfalls angegeben werden; falls es noch nicht existiert wird es für Sie erzeugt. Ob ein Arbeitsbereich für *Visual* oder ein *Makefile* erzeugt werden soll kann ebenfalls ausgewählt werden.

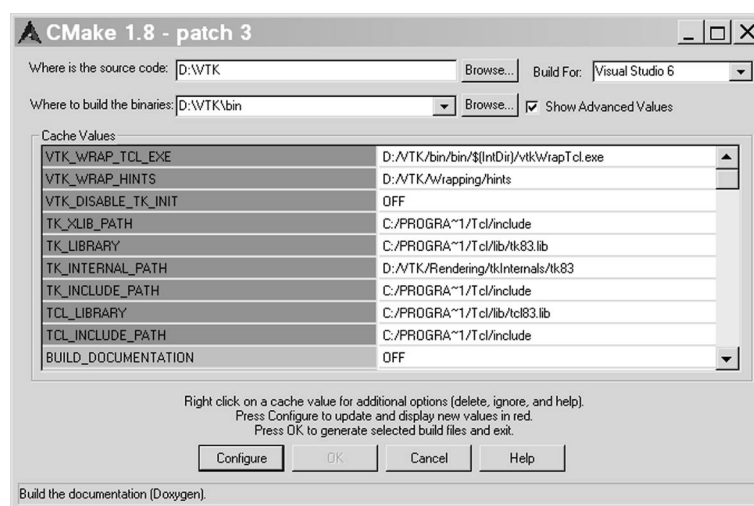


Abbildung 4.1: CMake unter *Microsoft Windows*

Cache-Variablen, die noch nicht richtig aufgelöst wurden haben einen Namen, der rot unterlegt ist. Durch Anklicken des Namens oder des Werts können Sie Schalter umlegen, oder neue Werte

definieren. Abbildung 4.2 zeigt die Entscheidung, ob die *tcl*-Bindung erzeugt werden soll.

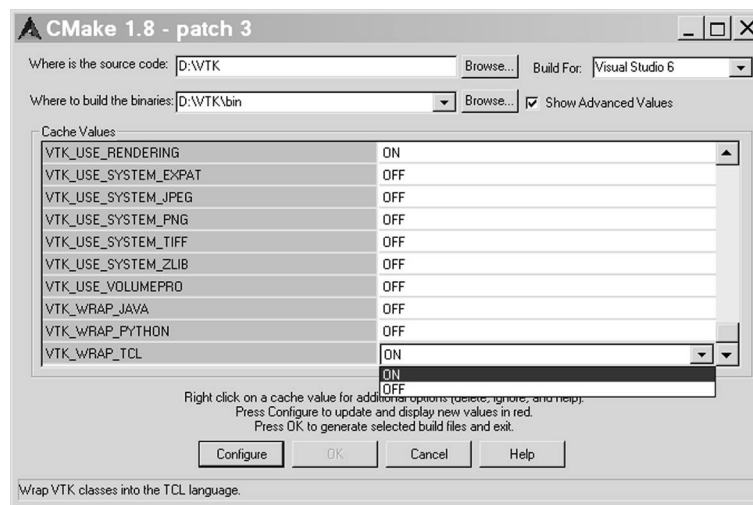


Abbildung 4.2: CMake unter Microsoft Windows

Nach einer Veränderung der Einstellungen empfiehlt es sich, durch die Taste *Configure* die Konfiguration anzustoßen. Je nach Einstellung erhalten Sie jetzt noch mehr Variablen. Verändern Sie diese so lange, bis alle Einträge grau hinterlegt sind. Dann können Sie mit *OK* das gewünschte Ergebnis erzeugen. Abbildung 4.3 zeigt diesen Zustand.

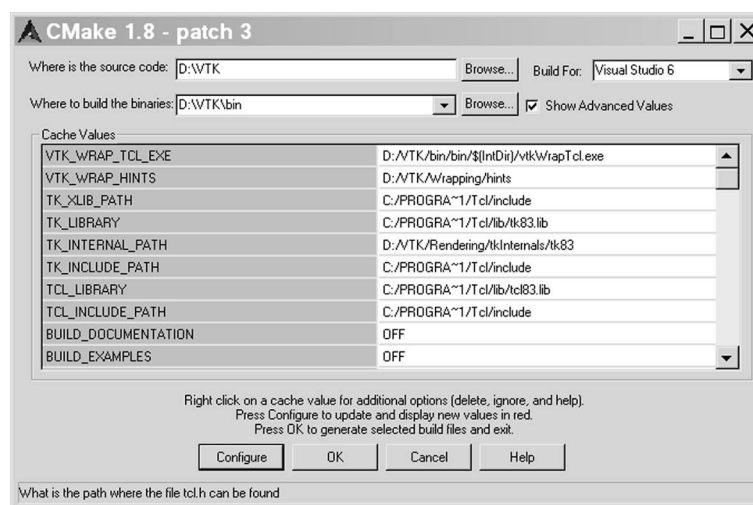


Abbildung 4.3: Alle Einträge sind auflösbar; durch *OK* kann jetzt der Arbeitsbereich oder der Makefile erzeugt werden

4.2 VTK mit Microsoft Visual C++ 6.0

Wir gehen in diesem Abschnitt davon aus, dass mit *CMake* ein gültiger Arbeitsbereich erstellt wurde. Falls Sie die Sprachbindungen für *Java*, *tcl* oder *Python* erzeugen wollen müssen Sie vorher noch das *jdk* oder entsprechende *Quellcode*-Installation der beiden Skriptsprachen installieren.

Im Verzeichnis, in dem Sie die Quellen ausgepackt haben und auch die *CMake*-Eingabedateien lagen existiert jetzt ein Arbeitsbereich, den Sie in *Visual* öffnen können. Legen Sie jetzt mit *Er-*

stellen — *Konfiguration festlegen* fest, welche Konfiguration sie bearbeiten möchten. Es bietet sich an, *ALL_BUILD* zu verwenden wie in Abbildung 4.4 dargestellt. Anschließend können Sie wie für andere Projekte die Quellcodes übersetzen und die Bibliotheken erzeugen. Danach finden Sie die Bibliotheken in Ordner `\bin`.

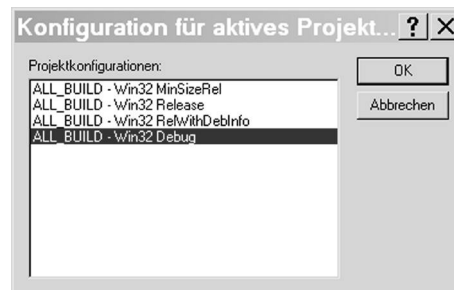


Abbildung 4.4: Festlegen der aktiven Konfiguration in Microsoft Visual

Tip:

Verwenden Sie nicht die Option *Alles neu erstellen*. Dadurch werden alle Eingabedateien `CMakeLists.txt` für *Cmake* gelöscht. Starten Sie in diesem Fall *Cmake* neu und erzeugen Sie einen neuen Arbeitsbereich!

Tip:

Wenn Sie den Wert für `VTK_USE_PATENTED` auf `ON` setzen, werden die in der Binär-Distribution nicht enthaltenen patentierten Algorithmen mit installiert!

4.3 VTK mit g++ auf cygwin oder LINUX

Für eine Installation auf *cygwin* oder *LINUX* benötigen Sie die entsprechende Quellcode-Datei, die Sie auf www.vtk.org finden. Falls noch nicht vorhanden, müssen Sie vorher *CMake* installieren.

Fall *curses* installiert ist, erzeugt die Installation von *CMake* ein Kommando *ccmake*. Dies stellt ein Oberfläche zur Verfügung, das für die verschiedenen Cache-Werte die aktuellen Einstellungen anzeigt und verändern lässt; analog zu der Darstellung in Abbildung 4.1. Eine terminalbasierte Version steht mit *cmake -i* zur Verfügung. Sind wir sicher, dass alle Einträge in `CMakeLists.txt` aufgelöst werden können, kann auch direkt *cmake* aufgerufen werden.

sind alle Einstellungen korrekt, dann haben wir jetzt einen *Makefile*. Mit *make* wird das Übersetzen und Erzeugen der Bibliotheken angestoßen. War dieser Vorgang erfolgreich, dann können wir mit *make install* die Installation abschließen.

Die größten Probleme bei der Installation machen verschiedene Compiler-Versionen. Allerdings sollten keine großen Probleme mit einem GNU-Compiler auftreten.

Tip:

Für die Einbindung der anderen Sprachbindungen ist es insbesondere bei *tcl* wichtig, dass *tcl/tk* als Quellcode-Installation vorliegt. Die Binär-Installation reicht nicht aus, um *VTK* zu installieren!

Tip:

Wenn Sie den Wert für `VTK_USE_PATENTED` auf `ON` setzen, werden die in der Binär-Distribution nicht enthaltenen patentierten Algorithmen mit installiert!

4.4 Eigene VTK-Projekte

`CMake` kann auch für das Erstellen von eigenen Projekten sinnvoll sein. Möglich ist, einen Makefile direkt zu erstellen oder den Arbeitsbereich wie oben beschrieben zu verändern. Gesetzt den Fall, es liegen auf einem PC beispielsweise sowohl eine Quellcode-Installation für *Visual* als auch für *cygwin* vor, dann können wir `CMake` dafür verwenden, den Makefile oder den Arbeitsbereich zu erstellen. Steht der Quellcode wieder in der Datei `pipeline.cpp`, dann können wir die folgende `CMakeLists.txt` verwenden:

```
PROJECT (pipeline)

INCLUDE (${CMAKE_ROOT}/Modules/FindVTK.cmake)
IF (USE_VTK_FILE)
  INCLUDE (${USE_VTK_FILE})
ENDIF (USE_VTK_FILE)

LINK_LIBRARIES(
  vtkRendering
  vtkGraphics
  vtkImaging
  vtkIO
  vtkFiltering
  vtkCommon
)

ADD_EXECUTABLE(pipeline pipeline.cpp)
```

Literaturverzeichnis

- [BB05] BENDER, MICHAEL und BRILL, MANFRED: *Computergrafik. Ein anwendungsorientiertes Lehrbuch*. Hanser, 2. Auflage, 2005.
- [Bri03a] BRILL, MANFRED: *Dokumentieren mit Doxygen*. Visualisierungslabor der FH Kaiserslautern, 2003.
- [Bri03b] BRILL, MANFRED: *Programmieren mit Cygwin und UNIX*. Visualisierungslabor der FH Kaiserslautern, 2003.
- [SMAL00] SCHROEDER, WILLIAM, MARTIN, KENNETH, AVILA, LISA und LAW, CHARLES: *The VTK User's Guide*. Kitware, Inc., 2000.
- [SML97] SCHROEDER, WILLIAM, MARTIN, KENNETH und LORENSEN, BILL: *The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*. Prentice Hall, 1997.