LINQ for Geometry - VORLÄUFIGES DOKUMENT

Implementierung der Half-Edge Datenstruktur zu Manipulation und Handling Dreidimensionaler Meshes insbesondere durch den Einsatz von LINQ und LAMBA Ausdrücken in Microsofts C#

Dominik Steffen

Erstbetreuer: Prof. Christoph Müller, Fakultät DM

Zweitbetreuer: Prof. Wilhelm Walter, Fakultät DM

15. Juli 2013

Inhaltsverzeichnis

1 Einleitung

1.1 Fragestellung

Ist es möglich die „Half-Edge Data Structure“ (kurz HES) in einer gemanagten Programmiersprache wie C# unter der Berücksichtigung von LINQ und Lambda Ausdrücken so zu implementueren, dass damit grundlegende Geometriemanipulation in der Computergrafik erfolgen kann?

1.2 Die Half-Edge Data Structure (kurz HES) als Algorithmus

1.2.1 Die Basis der HES

Verbindungen und Beziehungen in der HES

Doubly Connected Edge List

Die doubly connected edge list (kurz DECL) ist der verwendeten HES nicht unähnlich. Beide Datenstrukturen ähneln sich immens und haben doch ein paar wenige entscheidende Unterschiede. TODO

1.2.2 Speicherverbrauch im Gegensatz zu Face basierten Lösungen

1.2.3 Vorteile der HES

1.3 Aktueller Forschungsstatus

1.3.1 Probleme der aktuellen Forschung

1.4 Einführung zu LINQ in C#

1.4.1 Was ist LINQ in C#

1.4.2 „Highlevel“ LINQ in C#

1.4.3 „Lowlevel“ LINQ in C#

1.5 Einführung zu Lambda in C#

1.5.1 Was sind Lambda Ausdrücke in C#

1.5.2 Lambda Ausdrücke in Verbindung mit LINQ

2 Hauptteil

2.1 Gegenüberstellung nativer (OpenMesh.org) und gemanagter Implementierungen der Half-Edge Data Structure

2.2 Geschwindigkeitsunterschiede von nativem und C# Code

2.3 Die Vorteile in der Entwicklung von managed Code

2.4 Das Software Projekt LINQ For Geometry (LFG)

2.4.1 Warum die Programmiersprache C# ?

2.4.2 Furtwangen University Simulation and Entertainment Engine (FUSEE)

2.4.3 Grober Ablauf einer HES Initialisierung

UML Diagramme zum Initialisierungslauf

2.5 Der Import von Geometriedaten im „Wavefront Object“ Format

2.5.1 Warum das Wavefront Object Format

Face basierter Import - Edge basiertes Handling

2.5.2 Importer für das Wavefront Format

2.6 Implementierung und Funktion der Handler für die einzelnen Komponenten der HES

2.6.1 Beispiel eines Handler Konstruktes und seiner Implementierung

Listing 2.1: HandleHalf-Edge.cs - Variablen Deklaration des „Zeigers“

```
1        internal int _DataIndex;
```

Sie enthalten nur einen Index als Zeiger und sehr wenige Funktionen. Ein Handler speichert zur Laufzeit pro Instanz einen Index auf den realen Datensatz für den er einen Handle darstellt. Handler Structs gibt es für Edges, Half-Edges, Faces, FaceNormals, Vertices, VertexNormals und VertexUVs. Sie unterscheiden sich hierbei nur durch die Namensgebung der Structs und der Konstrukturen. Ein Handler Struct stellt eine implizite Konvertierung des Handlers, siehe Listing 2.2, in den Datentypen Integer (int) zur Verfügung.

Listing 2.2: HandleHalf-Edge.cs - Impliziter cast nach Integer

```
1        public static implicit operator int(HandleHalfEdge handle)
2        {
3            return handle._DataIndex;
4        }
```

Zusätzlich kann der Entwickler jederzeit abfragen ob der aktuell verwendete Handler schon als valide betrachtet werden kann. Hierzu Listing 2.3 betrachten. Ein Handler ist dann valide, wenn sein Index nicht kleiner als 0 ist. In dieser Arbeit werden einstweilen Handler initialisiert für die zum Zeitpunkt der Initialisierung noch kein Index zur Speicherung bereit steht. Diese vorerst nicht validen Handler werden dann mit dem Wert -1 initialisiert und sind somit zu diesem Zeitpunkt als nicht valide also nicht verwendbar zu betrachten. Um diese später im Programm zu benutzen, muss also noch der korrekte Index, meistens der Wert einer Count Funktion auf einer Liste eingefügt werden.

Listing 2.3: HandleHalf-Edge.cs - Is Valid?

```
1        public bool isValid
2        {
3            get { return _DataIndex >= 0; }
4        }
```

Eine Besonderheit der Handler ist die mit „internal“ gekennzeichnete Deklaration der Indizes. Siehe hierzu Listing 2.4. Durch das „internal“ C# Schlüsselwort können die Handler nur aus der jeweiligen gleichen Assembly angesprochen und verändert werden. Dies verhindert einen unbefugten oder unabsichtlichen Fremdzugriff von Außen. Während der Laufzeit wird also die Konsistenz der Datenstruktur in sich geschützt um so das Programm vor Abstürzen durch Zeiger Fehler zu schützen.

Listing 2.4: HandleHalf-Edge.cs - Deklarationen als internal

```
1        internal int _DataIndex;
```

> "Der interne Zugriff wird häufig in komponentenbasierter Entwicklung verwendet, da er einer Gruppe von Komponenten ermöglicht, in einer nicht öffentlichen Weise zusammenzuwirken, ohne dem Rest des Anwendungscodes zugänglich zu sein." [1]

Die vollständige Implementierung dieses Structs und aller sieben weiteren kann im Visual Studio 2010 Projekt auf dem diese Arbeit aufbaut unter folgender Verzeichnisstruktur betrachtet werden. „LinqForGeometry/LinqForGeometry.Core/src/Handles"

2.7 Pointer Container und ihre Rolle in LINQ For Geometry

2.7.1 Half-Edges Pointer-Container

Vertex Normal Handler

Vertex UV Handler

2.7.2 Edges Pointer-Container

2.7.3 Vertices Pointer-Container

2.7.4 Faces Pointer-Container

2.8 Der Geometry Teil in LINQ For Geometry

Hier Information über das Geometry Objekt und die darin enthaltenen Daten etc.

3 Schluss

3.1 Ergebnis der Arbeit

3.1.1 Wie weit ist LINQ For Geometry fortgeschritten

3.1.2 Welche Möglichkeiten zur Erweiterung durch eigenen Code bietet LINQ For Geometry

3.2 Zukünftige Entwicklungen und Ausblick auf die Verwendung von LINQ For Geometry

Listings

Literaturverzeichnis

[1] Microsoft C#-Referenz. http://msdn.microsoft.com, 2013.