1 VR-Beleg: Proposal

Max Große, Matr. 51539

1.1 Zielsetzung und Motivation

Ziel ist es, einen vollständigen Szenengraphen in C++ zu implementieren. Während bereits unmengen an fertigen Implementierungen eines Szenengraphen existieren, die bereits mit einer großen Menge an Funktionalitäten aufwarten, ist es für den Lerneffekt in meinen Augen dennoch förderlich, von Grund auf neu einen eigenen Szenengraphen zu entwickeln.

Mit Hilfe des eigenen Szenengraphens sollen dann diverse Techniken umgesetzt werden, welche durch den Einsatz einen Szenengraphen deutlich vereinfacht werden, wie etwa *Shadow-Mapping*, *Picking*, *Kollisionserkennung* und *Viewfrustum-Culling*.

Dabei soll der Schwerpunkt eher in Richtung Game-Engine liegen, das heisst ausschliesslich ein Betrachter/Benutzer sowie eine möglichst konsequente Trennung der Logik von der Graphik.

Mit dem Aufkommen von OpenGL 3.x bzw. inzwischen 4.x und dadurch den Wegfall der gewohnten Matrizen und Matrizenstacks, soll weiterhin versucht werden, soweit wie möglich auch hier auf diese zu verzichten und wie vorgesehen selber sämtliche Verwaltung und Verarbeitung von Transformationen zu übernehmen.

1.2 Geplante Features

Die wichtigsten Features, die implementiert werden sollen werden folgend kurz beschrieben.

1.2.1 Szenegraph

Erstellt werden soll ein Szenengraph welcher die Darstellung beliebiger Szenen ermöglicht. Der Szenengraph soll zum einen aus Knoten bestehen, welche ausschließlich eine Zusammenfassung von Kindknoten sowie deren lokale Transformation repräsentieren. Darzustellende Objekte befinden sich ausschliesslich in den Blättern, ein Anhängen von Kindknoten soll hier nicht möglich sein. Blätter wiederum enthalten eine Menge an Attributen, welche Eigenschaften wie etwa die Farbe enthalten und von mehreren Blättern geteilt werden sollen. Die Idee dahinter ist, dass einerseits eine Verkettung von Attribut-Knoten logisch nicht immer sinnvoll ist, und auf eine State-Machine hinauslaufen. Dadurch enthält der Graph entlang eines Pfades unter Umständen viele unnötige Knoten. Andererseits kann für jedes Blatt somit

direkt jedes Attribut erkannt und verändert werden, ohne dass eine erneute Traversierung oder gar Umordnung des Graphen notwendig wird.

1.2.2 Shadow-Mapping

Der Graph soll so gestaltet sein, dass eine Traversierung aufwärts, d.h. in Richtung Wurzel, ebenfalls möglich ist. Dadurch kann elegant an jeden Knoten eine Kamera gehängt werden, um die Szene aus dieser Position darzustellen. Somit kann problemlos für alle Lichter die Szene entsprechend in einen Tiefenpuffer gezeichnet werden, um mit dieser Information globale Schatten zu ermöglichen.

1.2.3 Picking und Kollisionserkennung

Durch die Verwaltung von einfachen Bounding-Boxes jedes Knotens kann effizient ein Strahlenschnitt (Picking) bzw. ein Bounding-Box-Schnitt (Kollisionserkennung) durchgeführt werden.

1.2.4 Viewfrustum-Culling

Ebenfalls kann durch die Bounding-Boxes effizient festgestellt werden, ob ein Knoten mitsamt aller Kindknoten überhaupt im sichtbaren Bereich der aktuellen Kamera liegt. Wenn dies nicht der Fall ist, brauchen entsprechend alle Kindknoten im aktuellen Frame nicht betrachtet werden.