Octree-Generierung: Visual Debugging mit Blender

Simon Daum
Lehrstuhl für Computergestützte Modellierung und Simulation
Technische Universität München
simon.daum@tum.de

Kurzfassung: Der Spatial Browser ist die Implementierung einer räumlichen Anfragesprache im dreidimensionalen Raum und wird unter anderem im Building Information Modeling eingesetzt. Eine wichtige Komponente des Spatial Browsers ist die Überführung dreiecksbasierter Körpern in eine Oktalbaum-Struktur. Um die Weiterentwicklung zu vereinfachen, wurde ein System entwickelt, das direktes visuelles Debugging der beteiligten Geometrie ermöglicht. Über plattformunabhängige Interprozesskommunikation wurde hierbei die Visualisierungs-software Blender mit der Hauptanwendung verknüpft. Das ausschließlich auf Open Source Software aufbauende System, erwies sich als performant und leistungsstark. Hierdurch konnte die Entwicklungszeit verkürzt und die Qualität der Hauptanwendung erhöht werden.

1 Einleitung

Um komplexe Software zu entwickeln, die fehlerfrei und performant arbeitet, bedarf es Wissen und Erfahrung. Die richtige Auswahl der Technologie und der eingesetzten Entwurfsmuster ist genauso entscheidend wie die fehlerfreie Implementierung einzelner Routinen. Mit dem Einzug virtueller Laufzeitumgebungen (Java, .NET) und der Verfügbarkeit entsprechender Frameworks (JUnit, NUnit) wurde die Infrastruktur für automatisierte Tests von Softwarekomponenten nochmals erweitert.

Trotz dieser Entwicklung kann während der Implementierung nicht vollständig auf manuelles Debugging verzichtet werden. Bildet die zu entwickelnde Software Entitäten im dreidimensionalen Raum ab, ist diese rein textuell basierte Herangehensweise jedoch nicht zielführend: Die menschliche Wahrnehmung kann zum Beispiel die Punktkoordinaten eines Dreiecks und eines Würfels nicht intuitiv zu einem inneren Bild verarbeiten, das Auskunft gibt, ob ein Schnitt dieser zwei Objekte stattfindet. Daraus folgt, dass räumliche Daten visualisiert werden müssen. Das gilt auch für Daten, die während einer Debugging Session gewonnen werden. Das hier vorstellte System zeigt auf, wie visuelles Debugging konzeptionell und letztlich konkret umgesetzt werden kann. Dazu werden ausschließlich Open Source Komponenten eingesetzt.

2 Der Spatial Browser

Building Information Modeling (BIM) unterstützt textuelle Abfragen, da hierfür die verfügbare Datenbank-Technologie herangezogen werden kann. So sind Abfragen wie „Selektiere alle Wände, die eine Dicke von 30 cm aufweisen.“ möglich. Möchte man aber räumliche Abfragen im dreidimensionalen Raum durchführen - um beispielsweise topologische Aussagen treffen zu können - ist die standardmäßige Datenbank- und BIM-Technologie dazu nicht in der Lage. Zur diesbezüglichen Erweiterung der Möglichkeiten im Building Information Modeling wurde am Lehrstuhl Computergestützte Modellierung und Simulation eine räumliche Anfragesprache (Borrmann, 2007) entwickelt. Die entsprechende Implementierung trägt den Namen Spatial Browser und ist eine in verschiedenen Ausprägungen verfügbare BIM-Software: als eigenständige Konsolen-Anwendung und als

eine Erweiterung für Autodesk Revit. Als Programmiersprache wird C# verwendet. Die Ausführung des Programmes erfolgt also in der Common Language Runtime, der virtuellen Maschine des .NET-Frameworks. Verfügbare topologische Operatoren der entwickelten Anfragesprache sind *disjoint, contains, inside, equals, meets, covers, coveredBy* und *overlaps.* Eine beispielhafte Anfrage lautet: „Selektiere alle tragenden Wände, die von Wand 3 berührt werden.“ Abbildung 1 zeigt das Query-Statement und die zurückgelieferte Selektion.

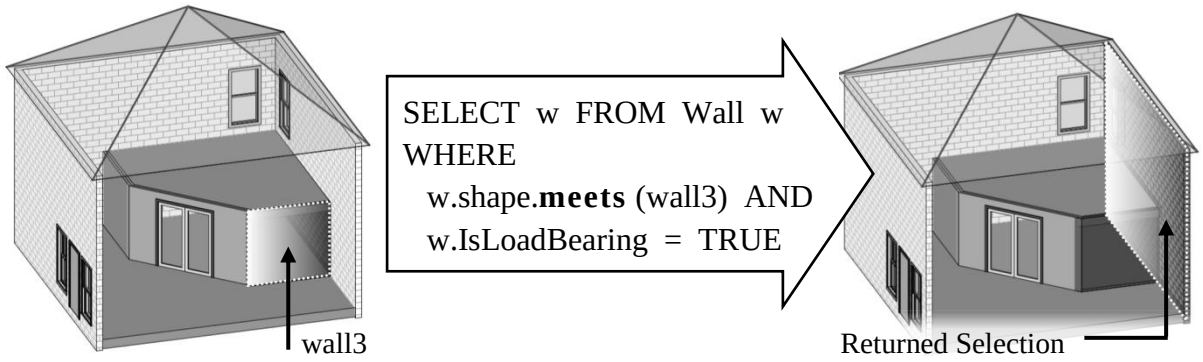

Abbildung 1:Topologische Anfrage im Spatial Browser

Die formale Definition der Operatoren geschieht über das 9-Intersection-Model (Egenhofer, 1991). In einer Matrix werden das Äußere, der Rand und das Innere zweier Objekte miteinander verschnitten. Aus der entstehenden Matrix kann, durch Vergleich mit Vorlagen, das topologische Prädikat bestimmt werden. Daraus folgt, dass die untersuchte Geometrie in eine Datenstruktur überführt werden muss, die Informationen über ihr Äußeres, ihren Rand und ihr Inneres enthält. Als passend erwies sich die Octree Struktur. Diese raumpartionierende, hierarchische Datenstruktur, deren Wurzel ein Würfel ist, wird bei Bedarf rekursiv verfeinert. Das Verhältnis der Kantenlänge eines Kindes zu seiner Elternzelle beträgt stets 1:2. So umfassen die acht Kinder, die folglich auch als Oktanten bezeichnet werden wieder denselben Raum wie ihre Elternzelle. Eltern- und Kind-Oktanten sind durch Zeiger miteinander verbunden und definieren so die Baumstruktur (Abb. 2). In genannten Anwendungsfall benötigt jeder Oktant außerdem ein Farbattribut, das seine relative Lage in Bezug auf die Hülle der untersuchten Geometrie ausdrückt: Oktanten im Inneren werden schwarz eingefärbt. Schneidet die Hülle der überführten Input-Geometrie einen Oktanten wird das Farbattribut dieses Würfels auf grau gesetzt. Weiß dient zur Kennzeichnung der im Äußeren liegenden Oktanten (Abb. 3).

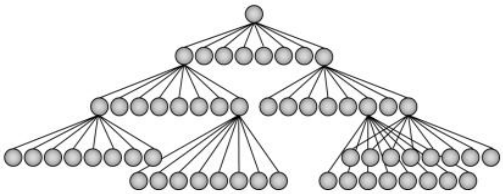

Abbildung 2: Zeigerstruktur eines beispielhaften Oktalbaums

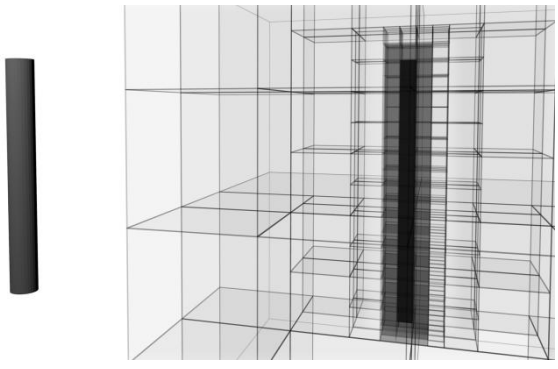

Abbildung 3: Geometrie und daraus erstellter, dreifarbiger Oktalbaum im Raum

Wurden die zwei zu untersuchenden Objekte in die beschriebenen dreifarbigen Oktalbäume überführt, kann nach Farbkombinationen gesucht werden um das 9-Intersection-Model zu befüllen und letztendlich eine topologische Aussage zu treffen.

Ein visuelles Debugging-Tool kann hier beispielsweise die Implementierung des Transfers von Boundary Represented Geometry (B-Rep) in dreifarbige Oktalbäume unterstützen. Dies ist nur einer von vielen möglichen Anwendungsfällen für visuelles Debugging im Spatial Browser. Zudem ist der Spatial Browser nur ein Vertreter geometriebasierter Konzepte und ihrer Implementierung. Kommerzielle als auch Open Source Tools bieten bis dato nur geringe Unterstützung dabei, geometrie-basierte Algorithmen effizient zu debuggen. Insbesondere die Visualisierungs-, Navigations- und Suchmöglichkeiten sind stark eingeschränkt (Data Display Debugger, 2012) und nicht mit einem 3D-Modelierungs-Tool wie Blender vergleichbar. Das hier vorgestellt Konzept ist ein genereller Lösungsansatz für das interaktive Debugging mit simultaner Dreiecks-basierter Geometrie-Visualisierung.

3 Blender

Die Open Source Software Blender ist für die Modellierung, Texturierung und Animation dreidimensionaler Körper ausgelegt und definiert sich selber als „3D Content Creation Suite“ (Blender Foundation, 2012). Aktuell ist die Version 2.63a der Software verfügbar. Neben Installationspaketen für die gängigen Betriebssysteme, ist auch der C-Quellcode zugänglich. Zur einfacheren Erweiterung und Automatisierung stellt Blender Python als Skriptsprache bereit. Mit ihr kann auf die darunterliegenden C-Strukturen zugegriffen werden. Abbildung 4 zeigt die Benutzeroberfläche.

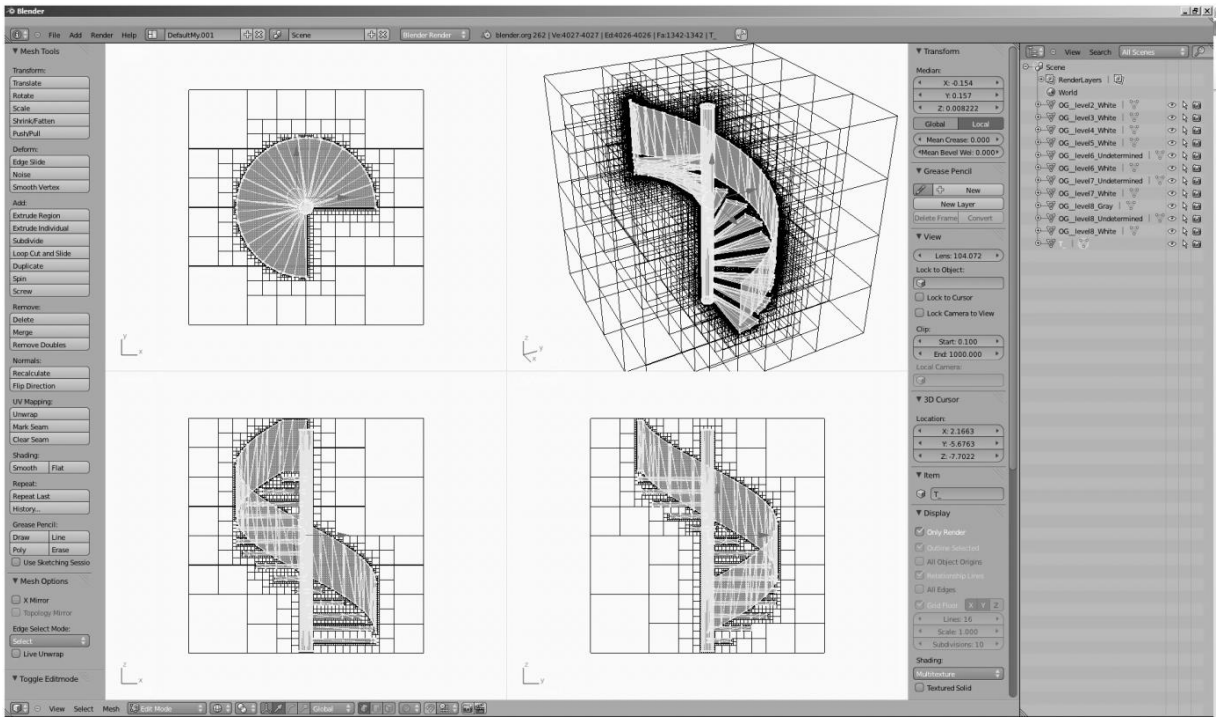

Abbildung 4: Benutzeroberfläche von Blender 2.63a mit geladener Ursprungs- und erstellter Oktalbaum-Geometrie

Eine Stärke von Blender liegt in der Erstellung, Bearbeitung und Visualisierung von Polygonnetzen. Dies deckt sich mit den Anforderungen an den visuellen Debugger des Spatial Browsers, da sowohl die Eingangs- als auch die Oktalbaum-Geometrie in Dreiecknetzen vorliegen. In der Szene in Abbildung 4 befindet sich die Ursprungsgeometrie (Wendeltreppe) als Dreiecksnetz sowie der daraus generierte Oktalbaum.

Um den Baum in Blender zu laden wurde auf die Importschnittstelle zurückgegriffen. Neben dem eigenen, proprietären Dateiformat können so eine Vielzahl anderer 3D-Formate in Blender eingelesen werden. Darunter befindet sich auch eine XML-Sprache zur Beschreibung von 3D Szenen und Objekten. Das Format trägt den Namen „Extensible 3D“ (X3D). Durch die gute XML Unterstützung in allen gängigen Programmiersprachen und auf Grund des klaren Aufbaus von X3D kann so mit geringem Aufwand ein Geometrie-Exporter aus der zu debuggenden Anwendung geschrieben werden um die Daten an Blender zu übergeben. Eine direktere Anbindung an die Hauptanwendung ist jedoch wünschenswert um Debugging-Aufgaben effizienter zu lösen. Ansonsten sind wiederkehrende, zeitraubende und fehleranfällige Aufgaben wie der Geometrie-Export und -Import auszuführen. Diese Tätigkeiten lenken den Entwickler stark vom eigentlich zu behebenden Problem ab. Eine unmittelbare Debug-Anbindung wurde durch die Entwicklung des Visual Debugger Systems basierend auf Interprozesskommunikation erreicht. Dies wird in Kapitel 4 näher beschrieben.

Blender bietet Möglichkeiten, die Echtzeit-3D-Darstellung anzupassen. Es stehen der BoundingBox-, der Wireframe-, der Solid- und der Textur-Modus zur Verfügung. Besonders der Wireframe-Modus ist im beschriebenen Anwendungsbeispiel wichtig: Die acht Kind-Oktanten nehmen denselben Raum wie ihr Eltern-Oktant ein. Durch diese Überlagerung wird der Baum in der Solid-Darstellung nur als ein einziger Würfel dargestellt. Im Wireframe-Modus kann, durch Selektion des Dreiecknetzes, außerdem eine flächenhafte, halb-transparente Darstellung, wie in Abbildung 5 dargestellt, erzielt werden.

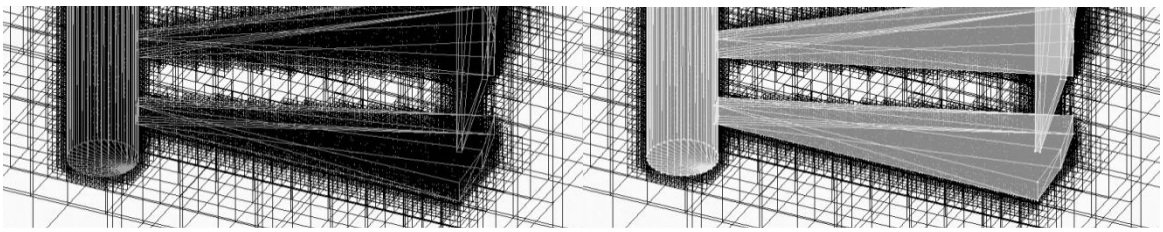

Abbildung 5: Visualisierungsmöglichkeiten in Blender

Insgesamt bietet Blender eine Vielzahl an Visualisierungs-, Navigations- und Selektionsmöglichkeiten. So sind die bekannten Grundfunktionalitäten zur Navigation im 3D Raum, wie Panning, Zooming und Kamera-Rotation, implementiert. Außerdem steht eine textuelle Suche nach Objekten über deren Namen zu Verfügung. Dies ist besonders nützlich, wenn mit einer Vielzahl von Objekten gearbeitet wird. Zur Strukturierung derart komplexer Szenen können Layer verwendet werden, denen Objektgruppen zugeordnet werden. Eine weitere Selektionsmöglichkeit besteht durch das Auswerten von Objektattributen. So können zum Beispiel alle weißen Oktanten selektiert werden. Das Programm erwies sich auch bei Szenen mit mehr als 250.000 Oktanten als reaktionsschnell und stabil. Mit dem hier vorgestellten Konzept lassen sich diese Funktionalitäten direkt zum visuellen Debugging nutzbar machen.

4 Visual Debugger

Um den Entwickler bei einer Debug Session im geschilderten Umfeld optimal zu unterstützen wurde der Visual Debugger entwickelt und in den Spatial Browser und Blender integriert. Das Gesamtsystem ist in Abbildung 6 dargestellt.

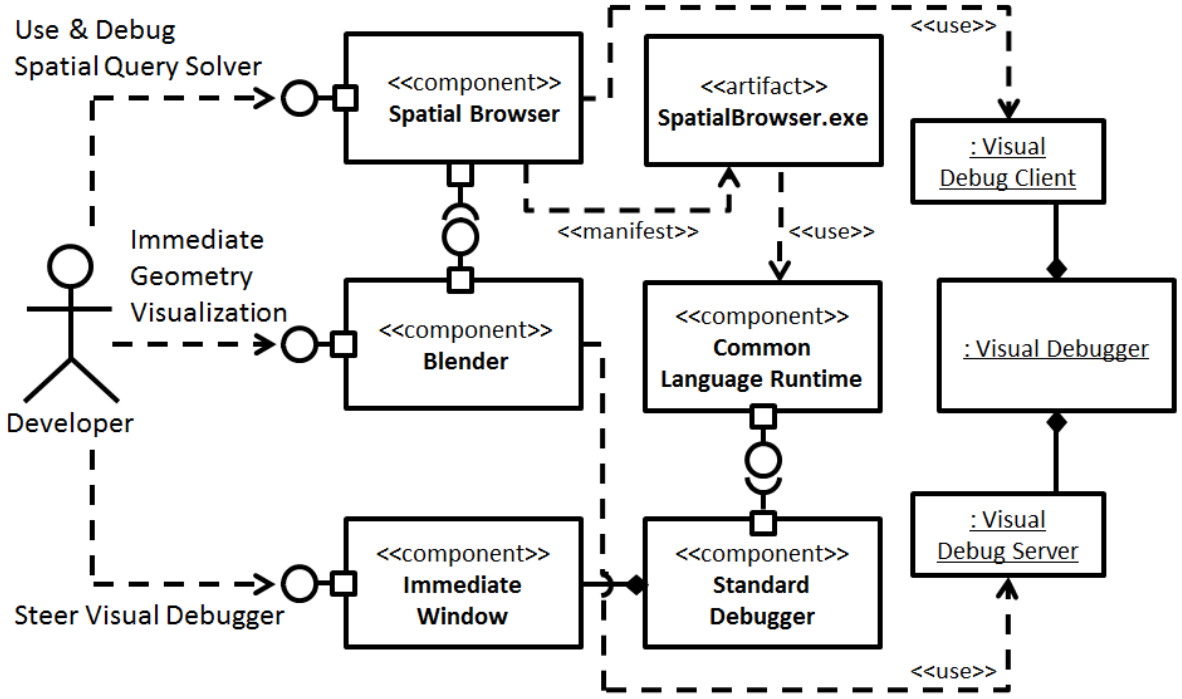

Abbildung 6: Der Spatial Browser mit Visual Debugging Funktionalität aus Sicht eines Entwicklers

Das System beschreibt die möglichen Use Cases aus Sicht eines Entwicklers. Diese Anwendungsfälle beinhalten das Stellen räumlicher Anfragen an den Spatial Browser. Als Ergebnis erhält der Entwickler eine Selektionsmenge bestehend aus Geometrieobjekten (siehe Abb. 1). Außerdem verbessert und erweitert der Entwickler das Spatial Browser System und muss folglich neue Funktionalitäten testen und debuggen. Dazu wird der Spatial Browser kompiliert und liegt danach als das Artefakt SpatialBrowser.exe vor.

Die mit dem entwickelten Visual Debugger verwirklichten Use Cases beinhalten die sofortige Darstellung geometrischer Daten während einer Debug Session im externen Programm Blender (Anwendungsfall „Immediate Geometry Visualization“ in Abb. 6). Um die Datenübermittlung zu steuern, bedient sich der Entwickler des Direkt-Fensters eines verbundenen Debuggers. Dieser Use Case wird „Steer Visual Debugger“ genannt und benötigt einen gestarteten Spatial Browser Prozess, der an einem Haltpunkt wartet. Dazu wird das SpatialBrowser.exe Artefakt in die Laufzeitumgebung (Common Language Runtime) des .NET-Frameworks geladen und ein Standard-Debugger mit dem Prozess verknüpft. An der zu debuggenden Stelle wird ein Breakpoint gesetzt und somit die Ausführung pausiert. Befindet sich das System in diesem Zustand kommt die Funktionalität des Visual Debuggers zu tragen. Er ermöglicht eine Kommunikation zwischen dem pausierten Spatial Browser Prozess und Blender.

Die zwei beteiligten Hauptklassen sind der Visual Debug Client und der Visual Debug Server. Wie in Abbildung 7 gezeigt, wird der Visual Debug Client innerhalb der Spatial Browser Komponente benutzt. In Blender ist die Visual Debug Server Klasse als Python Plug-In integriert. Der Aufbau dieser zwei Hauptklassen des Visual Debuggers und die damit verwirklichte Interprozesskommunikation werden im nächsten Abschnitt beschrieben.

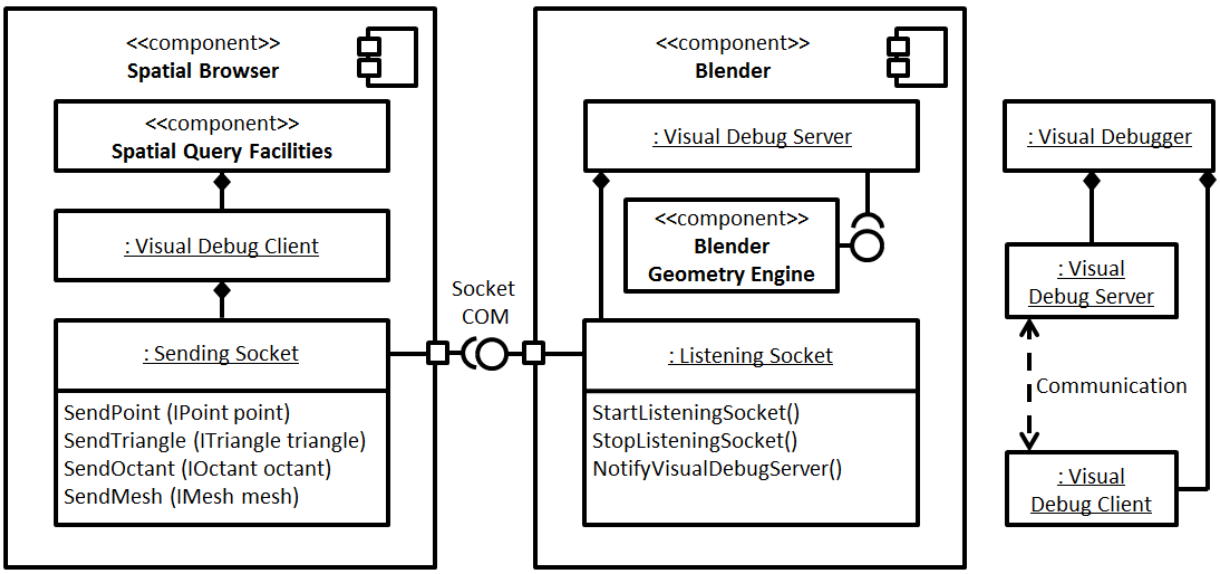

Abbildung 7: Die Komponenten des Visual Debugger Systems und ihre Einbindungen in den Spatial Browser und Blender

Der Visual Debug Client ist Teil des Spatial Browsers und wurde in C# umgesetzt. Er ist dazu ausgelegt während einer Debug Session der Hauptanwendung durch den Standard-Debugger aufgerufen und gesteuert zu werden. Ein Direkt-Fenster (Immediate Window) ist standardmäßig in Debug Umgebungen verfügbar und kann zu dieser Steuerung genutzt werden. Zu Beginn einer Debug Session wird der Standard-Debugger mit dem Prozess der Hauptanwendung (SpatialBrowser.exe) verbunden. Während der Prozess z.B. durch einen Breakpoint angehalten wurde, kann in das Direkt-Fenster Code eingegeben und unmittelbar ausgeführt werden. Für die Verwendung verfügbar sind alle Instanzen, die sich im Scope der momentan pausierten Ausführung befinden. Eine der typischen Anwendungen des Immediate Windows ist die Abfrage einer lokalen Variablen. Befindet sich im momentanen Geltungsbereich eine Variable, die Geometrie repräsentiert, kann somit auch auf diese zugegriffen werden. Der Spatial Browser definiert diesbezüglich die Geometrie-Interfaces IPoint, ITriangle, IOctant und IMesh.

Als Schnittstelle für den Spatial Browser zum Visual Debugger dient die statische Klasse VisualDebugClient mit je einer Send-Methode für jede vorkommende Geometrieart: IPoint, ITriangle, IOctant und IMesh. Diese Methoden können während des Debuggings im Direkt-Fenster durch Angabe des kompletten Namensraums und des Klassennamens live aufgerufen werden. Die Send-Methoden leiten über Socket-Kommunikation die als Parameter übergebene Geometrie an Blender weiter (Abb. 8).

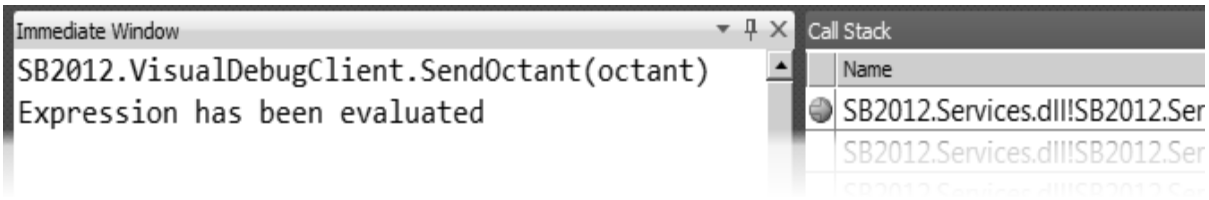

Abbildung 8: Steuerung des Visual Debug Clients über den Standard-Debugger. Über das Direkt-Fenster wurde ein Oktant versendet.

Der Visual Debug Server, die zweite Hauptklasse des Visual Debuggers, ist in Blender als Python Plug-In integriert. Sie ermöglicht es, ankommende Nachricht zu dekodieren und enthaltene Daten an die Geometry Engine in Blender weiterzuleiten. Außerdem implementiert das Plug-In ein User Interface zur Steuerung der Serverfunktionalität. Abbildung 9 zeigt diese Benutzeroberfläche und die über den Aufruf in Abbildung 8 übertragene Geometrie.

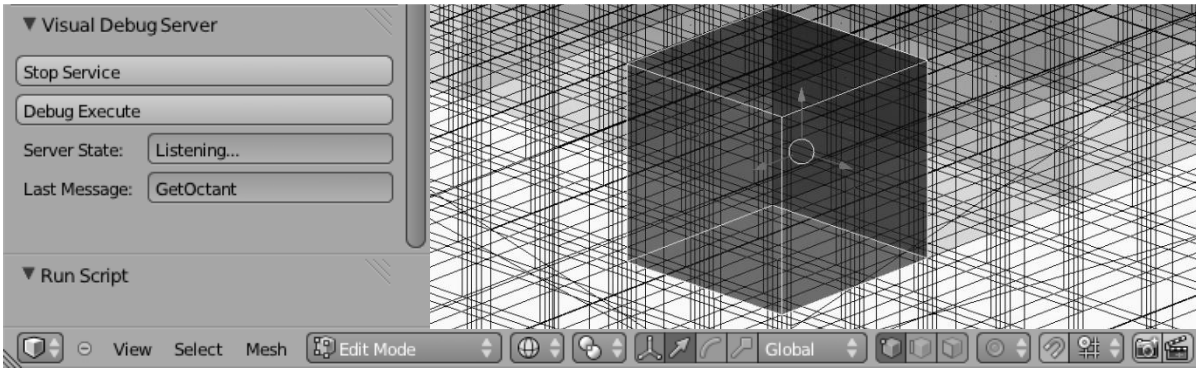

Abbildung 9: Benutzeroberfläche des Visual Debug Servers in Blender und aus simultan stattfindender Debugging Session übertragene Oktant-Geometrie

Über das User Interface lassen sich die in der ListeningSocket Klasse implementierten Socket-Funktionen aufrufen. Die Methode StartListeningSocket erzeugt einen neuen Thread in Blender, der für die Überwachung eines definierten Ports verantwortlich ist. Dazu wird ein neues Socket Objekt instanziiert und an den Port gebunden. Eingehende Daten an diesem Port werden ab jetzt durch den Server-Socket gelesen. Nach der Dekodierung des eingegangenen Nachrichtenstroms ist dieser in semantisch abgegrenzte Nachrichtenteile aufgesplittet worden. Der erste Teil beinhaltet die Art der geschickten Nachricht, zum Beispiele ITriangle oder IOctant. Der Anzeigename für die erhaltene Geometrie in Blender ist im zweiten Teil der Nachricht enthalten. Eine Liste an Punktkoordinaten und ein Transformationsvektor bilden das dritte und vierte Segment der Nachricht. Wird ein Mesh übertragen, ist außerdem eine Flächenliste, bestehend aus Punkt-Indizes notwendig, die dann als fünftes Nachrichten-Element verfügbar ist. Im Anschluss benachrichtigt die ListeningSocket Klasse den Visual Debug Server und veranlasst damit die Geometrieerstellung aus den erhaltenen Daten. Der Debug Server nutzt dafür die Blender Geometry Engine, die aus Punktkoordinaten und einer Indexliste Dreiecksnetze erstellt. Diese Liste ist bei separat gesendeten Dreiecken und Oktanten statisch und kann deswegen in Blender erzeugt werden, so dass auf ihre Übertragung verzichtet werden kann. Die durch die Geometrie Engine erzeugten Objekte werden der aktuellen Szene zugewiesen und erscheinen augenblicklich in der 3D-Ansicht.

Durch empirische Performance-Messungen konnte gezeigt werden, dass die Übertragung von einzelnen Punkten, Dreiecken und Oktanten bis zum Empfangen der Rückantwort vom Debug Server im Durschnitt innerhalb 10.000 Prozessorticks erfolgt. Für die interne Verarbeitung der Daten bis zu ihrer Visualisierung in Blender konnten ähnliche Werte

festgestellt werden. Die Fähigkeit des Visual Debugging Systems zur interaktiven Nutzung ist damit gegeben. Auch bei der Übertragung flächenreicher Dreiecksnetze wurden Antwortzeiten beobachtet, die aus Benutzersicht wünschenswert erscheinen: Eine Übertragung von ca. 100.000 Dreiecken dauerte im Mittel weniger als 0,5 Sekunden. Verdoppelt man diesen Wert wegen der zu addierenden Mesh-Erzeugungsdauer in Blender, liefert das System nach ca. 1,0 Sekunden eine 3D-Visualisierung des Netzes (Diagramm 1).

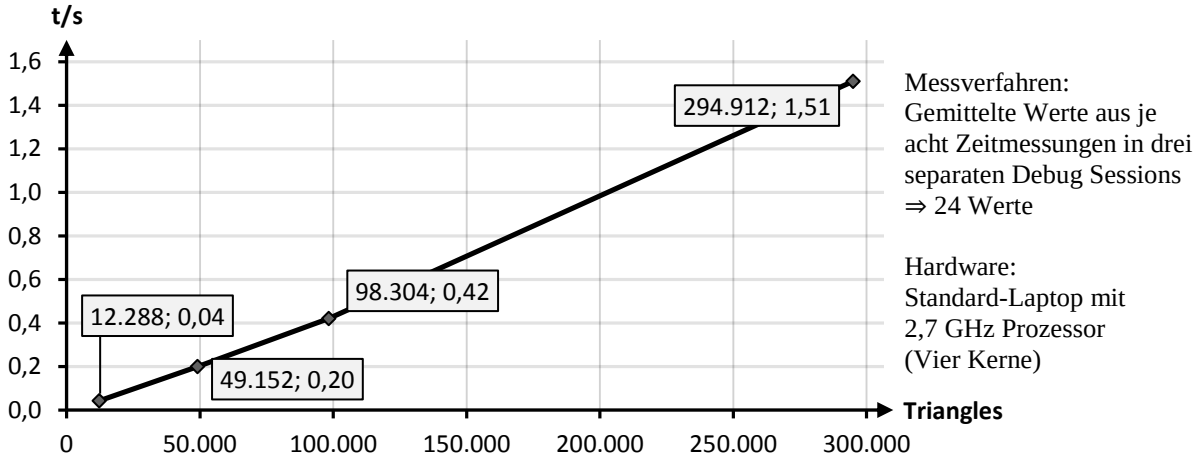

Diagramm 1: Messungen der Übertragungsgeschwindigkeit komplexer Dreiecksnetze im Visual Debugger System in Sekunden

5 Fazit

Das vorgestellte Konzept erläutert die Anbindung einer Visualisierungssoftware an eine Hauptanwendung während des Debuggings. Die verfügbare Referenzimplementierung verdeutlicht den Mehrwert, den ein Entwickler bei der Fehlersuche in geometrischen Algorithmen dadurch erfährt. Die hohe Performance des implementierten Systems erlaubt interaktives visuelles Debugging. Gekoppelt mit der ausgeprägten Visualisierungs-funktionalität im Zielprogramm Blender ist die Analyse beteiligter Geometrieobjekte robuster, einfacher und schneller als durch den manuellen, wiederkehrenden Import aufwendig erstellter 3D-Dateien. Die ausschließliche Verwendung von Open Source Software macht das System für Forschung und Lehre zusätzlich attraktiv. Der Quellcode des Visual Debuggers wird hierzu ebenfalls als Open Source veröffentlicht.

6 Literaturverzeichnis

Blender Foundation (2012). Blender Home - Startseite. Blender Foundation [Format: HTML, Zeit 26.06.2012, Adresse: http://www.blender.org/]

Borrmann, A., (2007). Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen u. räumlicher Datenbanken. Dissertation, Technische Universität München.

DDD Data Display Debugger (2012). DDT Home - Startseite. Free Software Foundation [Format: HTML, Zeit 01.08.2012, Adresse: http://www.gnu.org/software/ddd/]

Egenhofer, M. (1991), Reasoning about Binary Topological Relations. In: Proc. of the 2nd Symp. on Advances in Spatial Databases (SSD'91).