Effiziente numerische Feldsimulationen im Webbrowser durch hardwarenahe Implementierung auf der Grafikkarte

M.Sc. Christoph Lange, Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2 39106 Magdeburg, Deutschland, christoph.lange@st.ovgu.de,
M.Sc. Christian Bednarz, Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2 39106 Magdeburg, Deutschland, christian.bednarz@ovgu.de,
Prof. Dr.-Ing. Marco Leone, Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2 39106 Magdeburg, Deutschland, marco.leone@ovgu.de

1 Einleitung

In der heutigen Zeit sind numerische Feldsimulationen aus der Forschung und der Entwicklung komplexer elektronischer Geräte nicht mehr wegzudenken. Dennoch können sie bereits für vermeintlich einfache Probleme hinsichtlich der Rechenleistung und –dauer sehr aufwendig werden, was die Bereitstellung leistungsfähiger Rechentechnik notwendig macht. Des Weiteren sind kommerzielle Tools häufig sehr teuer und somit für Lehrzwecke, Privatpersonen oder Bildungseinrichtungen kaum zugänglich.
In dieser Arbeit wird ein Ansatz präsentiert, der es ermöglicht Feldsimulationen im Webbrowser durchzuführen. Die Berechnung erfolgt lokal durch die Ausnutzung der Grafikkarte. Am Beispiel der Finite-Differenzen-Methode werden dabei die wesentlichen Schritte zur Umsetzung erläutert. Abschließend wird der implementierte Algorithmus mit kommerzieller Simulationssoftware verglichen.

2 Grundlagen

Die Umsetzung eines Simulationsprogramms im Webbrowser erfordert zunächst die Programmierung der Hardware zur Beschleunigung der Berechnung. Die dafür verwendeten Bestandteile der Grafikkarte werden im Folgenden vorgestellt. Im Anschluss erfolgt die Herleitung der Gleichungen des Finite-Differenzen-Algorithmus.

2.1 Funktionsweise der Grafik-Hardware

Eine Grafikkarte berechnet ausgehend von Geometriedaten und Texturen Bilder zur Ausgabe auf dem Monitor. Für eine flüssige Darstellung mit 60 Bildern pro Sekunde ist eine sehr hohe Rechenleistung erforderlich. Dies wird dadurch realisiert, dass die Berechnungen parallel ausgeführt werden. Leistungsstarke Grafik-Hardware ist heutzutage in nahezu jedem Computer zu finden. Selbst OnBoard-Grafikchips (z.B in Laptops oder Mobiltelefonen) sind in ihrer Rechenleistung bei paralleler Verarbeitung CPUs weit überlegen. Diese zusätzliche Rechenleistung kann verwendet werden, um parallele Programmabläufe zu beschleunigen. Die Ansteuerung der Grafikkarte erfolgt über den Treiber, wobei hier OpenGL als Programmierschnittstelle verwendet wird. In dieser Arbeit wird die Grafikkarte mithilfe von Javascript und WebGL aus dem Webbrowser programmiert.
Abbildung 1 zeigt den Ablauf bei der Berechnung eines Bildes auf der Grafikkarte. Ausgehend von der Programmierschnittstelle (API) wird der auszuführenden Code an die beiden programmierbaren Einheiten (Vertex- und Fragment-Shader) übertragen. Die Geometriedaten werden in Form von Vertices an die Grafikkarte übermittelt. Nach einer Vorverarbeitung und Sortierung bildet der Vertex-Shader ausgehend von seiner Programmierung aus den Geometriedaten Flächen. Anschließend werden diese Flächen in Pixel unterteilt und die Farbwerte jedes Pixels

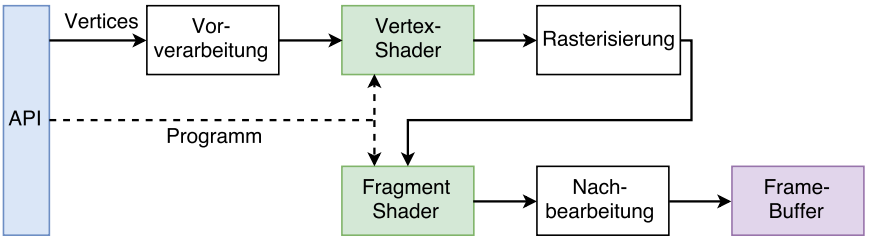

Abbildung 1: *Programmablauf zur Berechnung eines Bildes auf der Grafikkarte*

mithilfe des Fragment-Shaders berechnet. Das fertige Bild kann nun noch nachbearbeitet werden und wird in dem Frame-Buffer zwischengespeichert und anschließend ausgegeben. In Abschitt 4 wird gezeigt, wie mithilfe des Fragment-Shaders allgemeine Berechnungen durchgeführt werden können.

2.2 2D Finite-Differenzen-Algorithmus

In dieser Arbeit wurde der Finite-Differenzen-Algorithmus gewählt, um elektromagnetische Felder im Zeitbereich darzustellen. Dabei handelt es sich um ein numerisches Simulationsverfahren, bei dem die Maxwell'schen Gleichungen durch Differenzen approximiert werden. Die Herleitung und auch die Umsetzung als Parallelprogramm dieses Verfahrens sind ausführlich in der Literatur ([1], [2]) dokumentiert. Die Hardware-Beschleunigung durch Grafikkarten ist bereits in vielen kommerziellen Simulationstools implementiert. Eine direkte Umsetzung des Finite-Differenzen-Algorithmus mithilfe von OpenGL wurde in [3] vorgestellt. Diese Arbeit erweitert diesen Ansatz, um die Hardware-Beschleunigung auch im Webbrowser verwenden zu können.
Der Finite-Differenzen-Algorithmus kann aufgrund der Diskretisierung des Rechengebietes sehr einfach parallelisiert werden. Ein weiterer Vorteil ist, dass durch die unmittelbare Berechnung der Feldgrößen die Darstellung der Felder während der Simulation direkt möglich ist. Dies ermöglicht den Einblick in das Verhalten elektromagnetischer Felder innerhalb einer Problemstruktur.
Im Folgenden wird ein kurzer Überblick zur Herleitung des zweidimensionalen Finite-Differenzen-Algorithmus gegeben. Die Reduzierung auf zwei Dimensionen wurde vorgenommen, da die mit der Grafikkarte verarbeiteten Texturen ebenfalls zweidimensional sind. Somit reduziert sich der Programmieraufwand insbesondere bei der Erstellung der graphischen Benutzeroberfläche.
Zunächst werden nur die Rotationsgleichungen der Maxwell-Gleichungen des Freiraums betrachtet [4]:

$$\nabla \times \mathbf{E} = -\mu \frac{\partial \mathbf{H}}{\partial t} \tag{1}$$

$$\nabla \times \mathbf{H} = \varepsilon \frac{\partial \mathbf{E}}{\partial t} \,. \tag{2}$$

Bei der Betrachtung von zweidimensionalen Problemen wird angenommen, dass die Problemstruktur und die Anregung in z-Richtung unendlich weit und gleichförmig ausgedehnt ist. Somit können die Felder in diese Richtung auch als gleichförmig angenommen werden (mit $\frac{\partial}{\partial z} = 0$). Es folgt die Diskretisierung des Rechengebietes nach dem Yee-Schema [5] in Rechtecke mit

den Kantenlängen Δx und Δy. Dabei sind elektrisches und magnetisches Feld, wie in Abbildung 2 dargestellt, jeweils um die halbe Gitterbreite versetzt. Diese räumliche Verschiebung

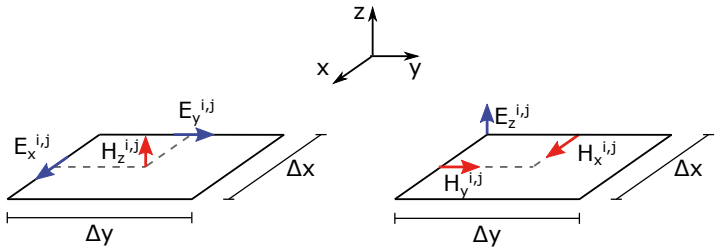

Abbildung 2: *Definition der Feldgrößen auf dem Yee-Gitter*

ermöglicht es die Feldkomponenten des elektrischen und magnetischen Feldes nacheinander zu berechnen. Zur Approximation auf dem Gitter wird folgende Notation verwendet:

$$E_z(x, y, t) \approx E_z(i\,\Delta x,\, j\,\Delta y,\, n\,\Delta t) = E_z^{i,j}\big|_n \,. \tag{3}$$

Die Ableitungen der Rotationsoperatoren in Gleichung (1) und (2) werden durch den zentralen Differenzenquotienten approximiert:

$$\left.\frac{\partial f}{\partial x}\right|_{x=x_0} \approx \frac{f(x_0+h) - f(x_0-h)}{2\,h} + \mathcal{O}\left(h^2\right) \,. \tag{4}$$

Der Approximationsfehler ist abhängig vom Quadrat der Gitterweite $h = \frac{\Delta x}{2}$. Man erhält zwei unabhängige Gleichungssysteme mit dem zur Ausbreitungsrichtung transversalen elektrischen Feld (TE):

$$H_x^{i,j}\big|_{n+\frac{1}{2}} = H_x^{i,j}\big|_{n-\frac{1}{2}} - \frac{\Delta t}{\mu^{i,j}}\,\frac{E_z^{i,j+1}|_n - E_z^{i,j}|_n}{\Delta y} \tag{5}$$

$$H_y^{i,j}\big|_{n+\frac{1}{2}} = H_y^{i,j}\big|_{n-\frac{1}{2}} + \frac{\Delta t}{\mu^{i,j}}\,\frac{E_z^{i+1,j}|_n - E_z^{i,j}|_n}{\Delta x} \tag{6}$$

$$E_z^{i,j}\big|_{n+1} = E_z^{i,j}\big|_n + \frac{\Delta t}{\varepsilon^{i,j}}\left[\frac{H_y^{i,j}\big|_{n+\frac{1}{2}} - H_y^{i-1,j}\big|_{n+\frac{1}{2}}}{\Delta x} - \frac{H_x^{i,j}\big|_{n+\frac{1}{2}} - H_x^{i,j-1}\big|_{n+\frac{1}{2}}}{\Delta y}\right], \tag{7}$$

bzw. magnetischen Feld (TM):

$$E_x^{i,j}\big|_{n+1} = E_x^{i,j}\big|_n - \frac{\Delta t}{\varepsilon^{i,j}}\,\frac{H_z^{i,j}\big|_{n+\frac{1}{2}} - H_z^{i,j-1}\big|_{n+\frac{1}{2}}}{\Delta y} \tag{8}$$

$$E_y^{i,j}\big|_{n+1} = E_y^{i,j}\big|_n + \frac{\Delta t}{\mu^{i,j}}\,\frac{H_z^{i,j}\big|_{n+\frac{1}{2}} - H_z^{i-1,j}\big|_{n+\frac{1}{2}}}{\Delta x} \tag{9}$$

$$H_z^{i,j}\big|_{n+\frac{1}{2}} = H_z^{i,j}\big|_{n-\frac{1}{2}} + \frac{\Delta t}{\mu^{i,j}}\left[\frac{E_x^{i,j+1}|_n - E_x^{i,j}|_n}{\Delta y} - \frac{E_y^{i,j+1}\big|_n - E_y^{i,j}\big|_n}{\Delta x}\right]. \tag{10}$$

Mit diesen Gleichungen ist es möglich für einen Zeitschritt Δt zunächst die Komponenten des magnetischen Feldes für das gesamte Rechengebiet zu berechnen und anschließend daraus die Komponenten des elektrischen Feldes zu berechnen.

Durch die Ortsabhängigkeit der Permeabilität und Permittivität können Materialien auf dem Gitter definiert werden. Die Anregung erfolgt durch die Addition einer Quelle an der Stelle p, q:

$$E_z^{p,q}|_n = E_z^{p,q}|_n + E_{\mathsf{Q}}|_n \; . \tag{11}$$

3 Umsetzung in WebGL

Anhand des Finite-Differenzen-Algorithmus wird die Umsetzung im Webbrowser erläutert. Dazu wird zunächst der Algorithmus an die Datenstrukturen der Grafikkarte angepasst. Im Anschluss erfolgt die Beschreibung des Programmablaufes im Webbrowser.

3.1 Programmierung der Grafikkarte

In einem Webbrowser besteht nur sehr eingeschränkt die Möglichkeit auf lokale Hardware zuzugreifen. Eine dieser Möglichkeiten ist die WebGL-Schnittstelle [6], mit deren Hilfe über einen OpenGL-fähigen Treiber die Grafikkarte angesprochen werden kann. Im Gegensatz zur Hardware-Beschleunigung mit einer Hochsprache, wie z.B. CUDA [7], ist hier nur eine direkte Programmierung möglich. Dies erfordert eine Anpassung des Algorithmus auf die Datenstrukturen der Grafikkarte.
Zur Umsetzung eines allgemeinen Berechnungsverfahrens auf der Grafikkarte müssen Daten bereitgestellt und abgerufen werden. Dazu werden Texturen verwendet, deren Farbwerte (Rot, Grün, Blau) als Gleitkommazahlen verwendet werden. Ausgehend von diesen Texturen können mithilfe der Grafikkarte neue Werte berechnet werden aus denen neue Texturen erstellt werden. Da eine Textur nicht gleichzeitig ausgelesen und beschrieben werden kann, werden zur Berechnung zusätzliche Texturen eingesetzt, um die Zwischenergebnisse zu speichern. Die Grafikkarte wird mit einer Textur initialisiert und berechnet für jeden Rechenschritt eine neue Textur, die am Ende ausgegeben wird.
Um mit den als Textur bereitgestellten Daten rechnen zu können wird der Fragment-Shader verwendet. Dieser wird für jeden Pixel der Geometrie aufgerufen und berechnet den Farbwert dieses Pixels. Abbildung 3 zeigt einen solchen Ablauf am Beispiel des Finite-Differenzen-Algorithmus. Die Berechnung des elektrischen und magnetischen Feldes erfolgt ausgehend von den Gleichungen (5)-(7).
Die Parameter für verschiedene Randbedingungen, wie z.B. Perfectly Matched Layer (PML) oder Perferct Elecric Conductor (PEC), Quellen und Materialien werden ebenfalls in Form einer Textur an die Grafikkarte übertragen und an entsprechender Stelle in den Programmablauf eingebunden. Die Amplituden der Feldkomponenten werden an Messpunkten erfasst und als Textur abgespeichert. Diese Messdaten werden nach dem Ablauf der Simulation an den Browser übertragen und dort aufbereitet.

3.2 Umsetzung im Webbrowser

Im Webbrowser wird mithilfe von Javascript eine Benutzeroberfläche (GUI) bereitgestellt. Diese ermöglicht die Konstruktion von zweidimensionalen Strukturen, die Anpassung der Parameter und die Auswertung der Ergebnisse.
In Abhängigkeit von der eingegebenen Problemstruktur und den Parametern werden mit Javascript entsprechende Shader- und Vertexprogramme erstellt und die Grafikkarte mit diesen programmiert. Anschließend werden die Texturen mit den Materialparametern, Randbedingungen und Quellen übergeben und Buffer zur Zwischenspeicherung von Ergebnissen initialisiert.

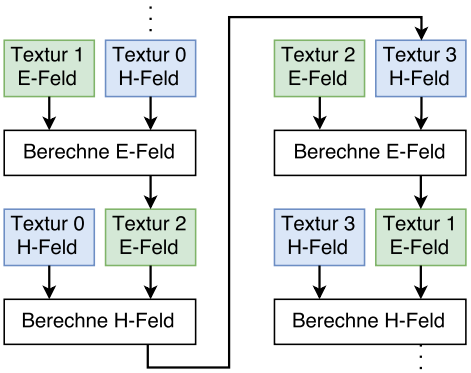

Abbildung 3: *Programmablauf des Finite-Differenzen-Algorithmus mithilfe von Texturen*

Mit einer Schleife werden die Feldwerte der einzelnen Zeitschritte $n\,\Delta t$ berechnet und als Texturen T_n zwischengespeichert. Die Auswertung erfolgt mit Feldkomponenten, die an zuvor definierten Gitterpunkten erfasst wurden und nach dem Durchlauf der Simulation an den Browser übergeben werden. In der Benutzeroberfläche erfolgt eine grafische Aufbereitung dieser Daten. Abbildung 4 zeigt diesen Ablauf als Blockdiagramm.

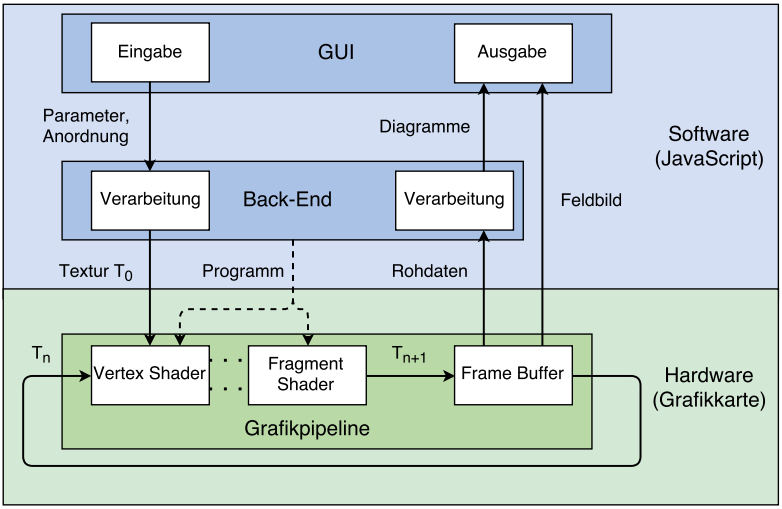

Abbildung 4: *Vollständiger Programmablauf des Simulationstools*

Während der Simulation kann aus den Feldkomponenten auf der Grafikkarte eine zwei- oder dreidimensionale Darstellung erstellt werden und als Animation direkt im Webbrowser angezeigt werden. Dies ermöglicht die direkte Beobachtung der transienten Vorgänge innerhalb der Struktur. Die Berechnungsgeschwindigkeit wird so angepasst, dass eine flüssige Darstellung

entsteht (mindestens 60 Bilder pro Sekunde). Dies reduziert die Berechnungsgeschwindigkeit, da die Grafikkarte die Bilder wesentlich schneller berechnen kann als dies für eine flüssige Darstellung erforderlich ist. Aus diesem Grund kann die Animation der Feldkomponenten ausgeschaltet werden. Somit ist die Berechnungsgeschwindigkeit nur von der internen Rechengeschwindigkeit der Grafikkarte abhängig.

4 Validierung anhand eines Beispiels

Anhand eines Beispiels soll der in WebGL implementierte Algorithmus mit einer kommerziellen Simulationssoftware verglichen werden. Dazu wird die in Abbildung 5 dargestellte Anordnung verwendet, um ein großes Rechengebiet mit einem Medienübergang von $\varepsilon_r = 1$ auf $\varepsilon_r = 4$ und eine Reflexion an einem perfekten Leiter (PEC) zu simulieren. Das Rechengebiet ist mit absorbierenden Rändern (PML) abgeschlossen und in 500×500 Zellen mit $\Delta x = \Delta y = 1\,\text{mm}$ unterteilt. Die Anregung erfolgt mit einem modulierten Gauß-Puls:

$$E_Q|_n = \mathrm{e}^{-\left(\frac{n\,\Delta t - t_0}{\tau}\right)^2} \sin(2\,\pi\,f\,n\,\Delta t)\,, \quad \text{mit} \quad t_0 = 0{,}2\,\text{ns}\,, \quad \tau = 50\,\text{ps}\,. \tag{12}$$

Es wird eine Frequenz von $f = 10\,\text{GHz}$ und einer Zeitdiskretisierung von $\Delta t = 2{,}22\,\text{ps}$ gewählt. Abbildung 6 zeigt die Amplitude der elektrischen Feldstärke bei Verwendung der TE-Gleichungen (5)-(7). Anhand des Zeitsignals des Empfängers erkennt man die Reflexion des Pulses an der Mediengrenze, die mit dem direkt verlaufenden Puls überlagert ist und die Reflexion an dem idealen Leiter.

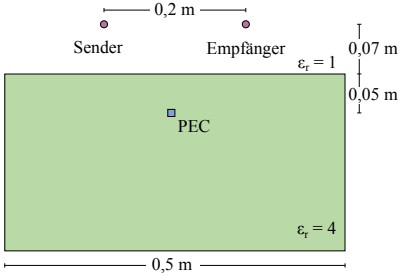

Abbildung 5: *Anordnung zur Validierung*

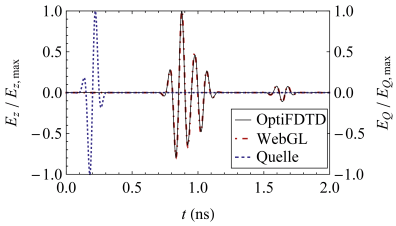

Abbildung 6: *Elektrische Feldstärke im Zeitbereich*

Zum Vergleich wurde das Simulationstool OptiFDTD [8] verwendet. Bei der Berechnung auf einem Computer mit Vierkern-CPU mit $3{,}4\,\text{GHz}$, $8\,\text{GB}$ RAM betrug die Rechenzeit mit OptiFDTD $36{,}8\,\text{s}$. Im Vergleich dazu betrug die Rechenzeit des in WebGL implementierten Algorithmus mit einer Nvidia Geforce-Grafikkarte der Serie 9 $4{,}36\,\text{s}$. Dieser ist somit achtmal schneller als die kommerzielle Simulationssoftware ohne Hardware-Beschleunigung.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde am Beispiel eines Finite-Differenzen-Algorithmus die Verwendung der lokalen Grafik-Hardware zur Beschleunigung von Berechnungen im Webbrowser vorgestellt. Es hat sich gezeigt, dass sich dadurch die Berechnungszeit auch im Vergleich zu kommerzieller Software stark reduzieren lässt. Mit diesem Verfahren ist es möglich auch komplexe Simulationssoftware mit dem Webbrowser als Plattform nahezu überall verfügbar zu machen.

Die intuitive Benutzung im Webbrowser und die effiziente Nutzung der lokalen Hardware ermöglichen es einem weiten Benutzerkreis Einblicke in das Verhalten von elektromagnetischen Feldern zu erlangen, z.B. auch in der Lehre.
Der Algorithmus kann auf dreidimensionale Strukturen erweitert werden. Dazu wird der Berechnungsraum in einzelne Ebenen zerlegt, die zu Texturen zusammengefasst werden. Die Berechnung erfolgt analog zu dem oben beschriebenen Verfahren. Des Weiteren ist es möglich andere parallelisierbare Algorithmen umzusetzen, die zur Simulation von elektromagnetischen Problemstellungen verwendet werden können.

Literatur

[1] Kunz, Karl S. ; Luebbers, Raymond J.: *The finite difference time domain method for electromagnetics*. CRC press, 1993

[2] Taflove, Allen ; Hagness, Susan C.: *Computational electrodynamics*. Artech house, 2005

[3] Adams, Samuel ; Payne, Jason ; Boppana, Rajendra: Finite difference time domain (FDTD) simulations using graphics processors. In: *DoD High Performance Computing Modernization Program Users Group Conference, 2007* IEEE, 2007, S. 334–338

[4] Jackson, John D.: *Classical Electrodynamics*. John Wiley & Sons, 1998

[5] Yee, Kane S. u. a.: Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. In: *IEEE Trans. Antennas Propag* 14 (1966), Nr. 3, S. 302–307

[6] Matsuda, Kouichi ; Lea, Rodger: *WebGL programming guide: interactive 3D graphics programming with WebGL*. Addison-Wesley, 2013

[7] Nvidia Corporation: *Compute Unified Device Architecture (CUDA), Version 7.5*. `http://www.nvidia.com/object/cuda_home_new.html`, 2015

[8] Optiwave Systems Inc.: *OptiFDTD (32-Bit Freeware), Version 12.0*. `http://www.optiwave.com/`, 2014