Sich bewegende und kommunizierende Agenten

Aufgabenbeschreibung:

Implementieren Sie ein Multi-Agenten-System. Entscheiden Sie sich mit Hilfe der Tutoren für eine Pragrammiersprache und eine Parallelisierungstechnologie.

Für diese Aufgabe wäre es ideal, wenn Sie Zugang zu notwendiger Hardware (Cuda bzw. Mehrkernrechner) hätten. Ein Mehrkernrechner mit einem PVM-Setup und Nvidia-Grafikkarte (muss noch geklärt werden) könnte eventuell bereitgestellt werden.

Modellbeschreibung:

Auf einer zweidimensionalen Ebene befinden sich zwei Klassen von Agenten. Die erste Klasse bewegt sich nicht und wird vor Simulationsbeginn als Anfangsbedingung auf dem Gebiet positioniert. Jeder dieser Agenten beinhaltet einen Vektor, der eine Bewegungsrichtung und -geschwindigkeit definiert. Dieser Vektor kann während der Simulation von in der Nähe befindlichen Agenten der zweiten Klasse gelesen werden und dient als "Wegweiser" für deren Bewegung im nächsten Zeitschritt.

Jeder bewegliche Agent hat eine Anfangsgeschwindigkeit, eine Anfangsrichtungrichtung und ein Sichtfeld mit Radius r. Tritt ein Agent erster Klasse in das Sichtfeld eines beweglichen Agenten, ändert dieser seinen Bewegungsvektor für den nächsten Zeitschritt.
Sei v_t der Bewegungsvektor zum Zeitpunkt t und liegt ein Wegweiser mit Vektor w im Sichtfeld, also innerhalb des Radius r, dann ändert der Agent seinen Vektor auf v_t+1=(v_t+w)/2. Liegen zwei Wegweiser im Sichtfeld, dann gilt v_t+1=(v_t+w_1+w_2)/3 usw.

Jeder Wegweiser beinhaltet also zwei Koordianten für seine Position und zwei für seinen Richtungsvektor. Ebenso die Agenten der zweiten Klasse.

Der Sinn dieses Modells ist es, durch eine bestimmte Positionierung der statischen Agenten eine Strömungsdynamik für die beweglichen Agenten zu erzeugen.
Eine typische Anfangsbedingung wäre z.B eine gleichmässige zufällige Verteilung aller Agenten auf dem Einheitskreis um den Punkt (0,0). Den Agenten der ersten Klasse (mit Position (x,y)) wird dann der Richtungsvektor D(pi/2)*(x-0,y-0) zugewiesen. Wobei D(phi) die Drehmatrix um den Winkel phi ist. Die Agenten der zweiten Klasse erhalten einen zufälligen initialen Bewegungsvektor, dessen Länge z.B. zwischen 0 und 0.5 liegt. Dadurch sollten sich die beweglichen Agenten annähernd im Kreis um den Mittelpunkt (0,0) bewegen.
Werden während der Simulation die Richtungsvektoren der Wegweiser verändert, sollte sich das Strömungsverhalten an die neuen Bedingungen anpassen.

Parallelisierungsstrategie:

Da der Kommunkationsaufwand bei der Parallelisierung relativ groß ist (die Agenten müssen ständig Abstandsberechnungen durchführen), kann es sein, dass sich ein Zeitgewinn erst bei einer sehr großen Anzahl an Agenten einstellt.

Statten Sie die Agenten der zweiten Klasse jedoch mit der Fähigkeit aus, vorausdenkend zu berechnen, können Sie zeitaufwändige Synchronisationen (mit den Agenten der ersten Klasse) vermeiden. Merkt sich ein Agent z.B. die Koordinaten und Richtungsvektoren aller Wegweiser im Umkreis von 3*r Längeneinheiten, so muss eine Synchronisation nur noch jeden zweiten Zeitschritt erfolgen.
Dabei müssen Sie einerseits auf den verfügbaren Speicherplatz Rücksicht nehmen und gleichzeitig darf die Bewegungsstrecke nicht größer als 2*r sein, da sonst notwendige Wegweiser im Zwischenspeicher des Agenten fehlen.

Die Parallelisierungsstrategie hängt aber stark von der verwendeten Methode und den geometrischen Bedingungen (z.B. Dichte) ab!

Aufgabenstellung:

Wählen Sie ein Kombination aus Programmiersprache und Parallelisierungsstrategie!
Zur Kontrolle sollten Sie eine sehr einfache grafische Ausgabe erzeugen, oder für jeden Zeitschritt (zur späteren Visualisierung) eine nummerierte Textdatei mit den Koordinaten der beweglichen Agenten ausgeben.
Vergessen Sie nicht auf eine Beschreibung Ihres Codes und eine Dokumentation.

Programmiersprache	Parallelisierungsmethode
MATLAB + Schnittstelle (www.mb.hs-wismar.de/cea/dp/)	PVM (www.csm.ornl.gov/pvm)
MATLAB + Schnittstelle (www.nvidia.com/...)	Cuda (www.nvidia.com/...)
C/C++	PVM (www.csm.ornl.gov/pvm)
C++	Cuda (www.nvidia.com/...)
C++	Threads (z.B. pthreads)
Java	JPVM (www.cs.virginia.edu/~ajf2j/)