Lucas Euzébio Machado

Parallel Algorithms for Multicore Game Engines

TESE DE DOUTORADO

DEPARTAMENTO DE INFORMÁTICA
Postgraduate program in Informatics

Rio de Janeiro
March 2010
Lucas Euzébio Machado

Parallel Algorithms for Multicore Game Engines

TESE DE DOUTORADO

Thesis presented to the Postgraduate Program in Informatics of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Doutor em Informática

Advisor: Prof. Bruno Feijó

Rio de Janeiro
March 2010
Lucas Euzébio Machado

Parallel Algorithms for Multicore Game Engines

Thesis presented to the Postgraduate Program in Informatics, of the Departamento de Informática do Centro Técnico Científico da PUC-Rio, as partial fulfillment of the requirements for the degree of Doutor.

Prof. Bruno Feijó
Advisor
Departamento de Informática — PUC–Rio

Prof. Noemi Rodriguez
Departamento de Informática — PUC–Rio

Prof. Marcelo Dreux
Departamento de Engenharia Mecânica — PUC–Rio

Prof. Esteban Gonzalez Clua
Universidade Federal Fluminense — UFF

Prof. Cesar Tadeu Pozzer
Universidade Federal de Santa Maria — UFSM

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico da PUC–Rio

Rio de Janeiro, 19/3/2010
Lucas Euzébio Machado

Lucas Machado graduated in Information Systems at PUC - Rio. He maintained a CAPES scholarship during his Masters and PHd degree, where he developed work applied to the area of electronic games. Currently he works teaching game programming and computer science.

Bibliographic data

Machado, Lucas

Parallel Algorithms for Multicore Game Engines / Lucas Euzébio Machado; advisor: Bruno Feijó. - 2010.

v., 70 f: il. ; 29,7 cm


Inclui bibliografia.


CDD: 004
To my parents.
Acknowledgments

Special thanks to my advisor Bruno Feijó for basically everything and to my colleague and friend Rodrigo Martins for the free consulting.
Resumo


Esse tese apresenta diversas técnicas sobre tecnologia paralela em jogos eletrônicos. A tese inicia apresentando diversas arquiteturas possíveis para um motor de jogos. Uma nova arquitetura é proposta, mais flexível e adequada para processadores do futuro que terão um grau maior de paralelismo. Em seguida, uma nova técnica para processar uma octree, uma estrutura de dados clássica da computação gráfica, é apresentada. As últimas técnicas apresentadas são relacionadas a detecção de colisão. Novas técnicas para processamento de grids hierárquicos e balanceamento de detecção de colisão em um conjunto de objetos são apresentadas.

**Palavras–chave**

Comutação Paralela; Programação de Jogos Paralela; Motores de Jogos Paralelos;
Abstract


This thesis presents several techniques about parallel technology on electronic games. The thesis begins presenting several possible architectures for a game engine. A new architecture is presented, more flexible and adequate for the processors of the future that will have a higher level of parallelism. Following, a new technique for processing an octree, a classic data structure for computer graphics, is presented. The last techniques presented are related to collision detection. New techniques for processing hierarquical grids and balancing collision detection on a set of objects are presented.

Keywords

Parallel Computing; Parallel Game Programming; Parallel Game Engines;
List of Figures

1.1 Project Gotham Racing Threads (source:[Dawson06]) 13
1.2 Kameo’s Threads - a more balanced task division (source:[Dawson06]) 13
1.3 An octree 15

2.1 A Single Thread Game Loop 18
2.2 A Synchronous Parallel Function Architecture running on 3 cores 19
2.3 An Asynchronous Parallel Function Architecture running on 3 cores 20
2.4 A Data Parallel Architecture running on 3 cores 21
2.5 A Pipeline Architecture 22
2.6 A Fully Parallel Architecture running on 3 cores 23

3.1 Example of spatial division with quadtree. Nodes that are relevant for the frustum are marked by (*) and irrelevant ones have dashed branches. 26
3.2 A possible work distribution for the quadtree of figure 3.1 at level 1. 28
3.3 The split level of the A2SSB algorithm for p processors revealing the two-step nature of the proposed algorithm 29
3.4 nodeList being used by the Node Processing Step One and Two 32
3.5 Intel quad-core processors. In the present work, tests are made on the Intel Core 2 Extreme Quad-core processor (a) [Intel09] 33
3.6 Only an uniform octree partition (a) is allowed by the proposed algorithm at step two. 33
3.7 Octree nodes as boxes. The numbers are the child IDs. 33
3.8 The translation matrix that is used for finding all brothers of an octree node 34
3.9 Examples of trends affecting the split level d and the total idle time. At first the trend is set as true and is constantly increasing d. This reduces the total idle time until we arrive at a moment where the increase of d also increased the total idle time, this results in the trend being set to false and we repeat the process, but now the trend constantly decreases d. 37
3.10 Performance analysis of octree node processing algorithms for an octree with 299,593 nodes and running on an Intel Core 2 Extreme Quad-core computer. 41
3.11 L2 cache misses for the proposed algorithm A2SSB (ALGO1 in Table of figure 3.10) 41
3.12 L2 cache misses for the version of A2SSB without a cache friendly strategy (ALGO2 in Table of figure 3.10) 42
3.13 Example of counting sort 46
3.14 Simple cases of the n-dimensional resource data array R. 47
3.15 The merged resource data array M from P_1..P_4 48
4.1 2D Grid dividing the space into equal sized cells.
4.2 A grid with small cell size and big objects.
4.3 A grid with big cell size and small objects.
4.4 An object may be colliding against objects in the southwest cell.
4.5 Simple Parallel Collision Detection.
4.6 The processing of the first object for each processor.
4.7 A better selection for each processor to handle.
4.8 Saving the parallel collision response results.
4.9 Parallel Collision Count Table.
4.10 Parallel Collision Count with Padding Results.
4.11 Cache line loading of 2 elements of a buffer.
4.12 Parallel Collision Count with Memory Alignment using 10,000 objects.
4.13 Parallel Collision Detection with Memory Alignment using 10,000 objects.

5.1 Results for the octree node processing with 299,593 nodes and 4 processors.
...Embrace Change

Kent Beck.