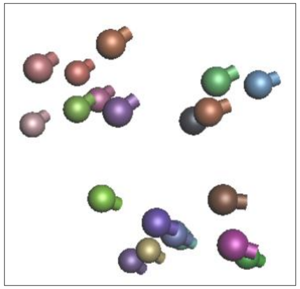
Cheat Detection Processing: A GPU versus CPU Comparison

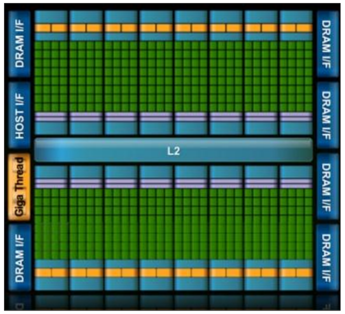
Håkon Kvale Stensland, Martin Øinæs Myrseth, Carsten Griwodz, Pål Halvorsen

Simula Research Laboratory / University of Oslo

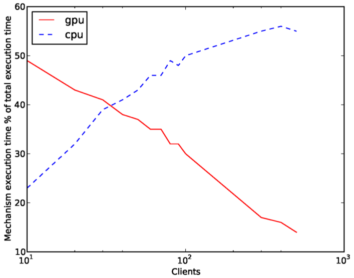
This presentation will give an introduction to local cheat detection processing using GPUs.



Cheat Detection and our example game



Nvidia Graphics Processing Units



Results and Conclusion

Cheating in computer games

- Multi-player gaming has experienced an amazing growth over the last decade, and cheating is the most prominent case of malicious behavior.
- Many on-line multi-player games still suffer from excessive cheating in one form or another. In many cases, the existence of the cheating is hard to prove.
- Most games that have implemented in-game physics use it as a major part of the game-play. In-game physics is therefore a very likely part of a game to be exploited.
- Our goal in this paper is to determine if graphics processing units (GPUs) can offload cheat detection mechanisms in a client-server based game system.

Classification of Cheats

Game level cheats:

- Breaking rules in the game and misusing features
- No modifications to the game client needed

Application level cheats:

- Modifications to game client or the operating system
- Example: Aimbots, reflex enhancers and farming bots

Protocol level cheats:

- Changing or delaying contents of packets
- Will appear only as high latency to other players and the central server

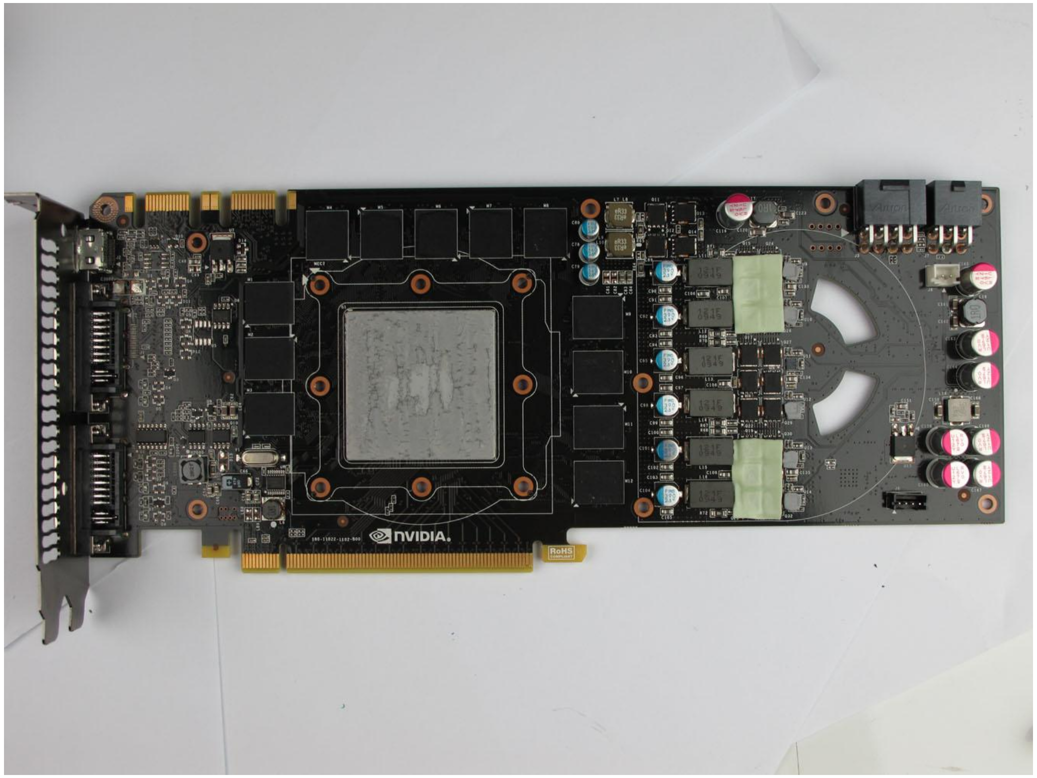
Infrastructure level cheats:

- Modifications to drivers, libraries, hardware, network etc.
- Example: Don’t render textures, and see through objects

Existing Cheat Detection Mechanisms

- **Most existing solutions run on the client side:**
  - Applications running on the client-side are also a likely target for exploits.
- **Deployed Detection Mechanisms:**
  - **VAC:** Valve Anti-Cheat is integrated in Steam. Can not detect content hacks, i.e. transparent textures used by the driver.
  - **Punkbuster:** 3rd party tool which search memory on the client side for known exploits using signatures.
  - **Warden:** Developed by Blizzard. Scans the Warcraft game memory space. Can also search out-of-process for known cheating drivers.

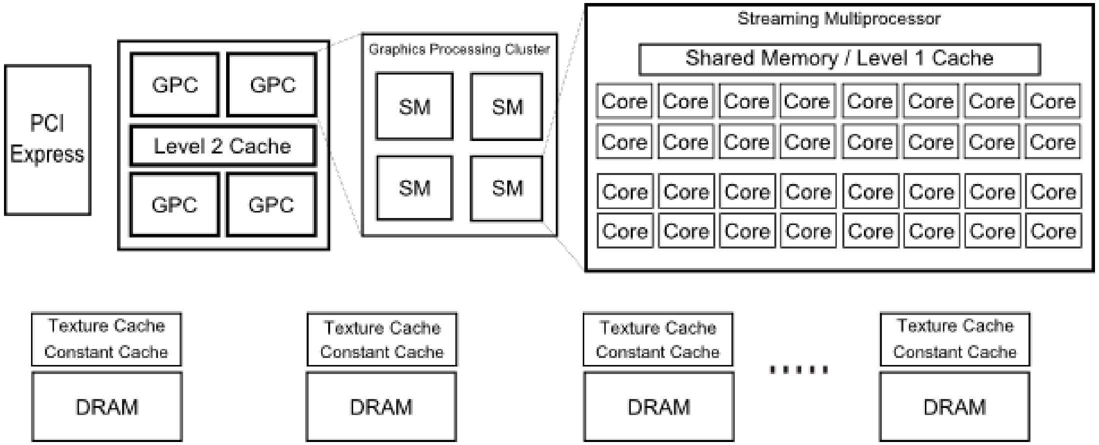
Nvidia GeForce Graphics Processors



- Nvidia GeForce GTX 480
- Based on the latest generation GPU, codenamed GF100 (Fermi)
- 3 billion transistors on TSMC 40nm (Made in Taiwan ☺)
- 480 Processing cores (CUDA Cores) at 1401 MHz
- 1536 MB Memory with 177,4 GB/sec of bandwidth.
- 1344.96 GFLOPS (FMA) of computing power

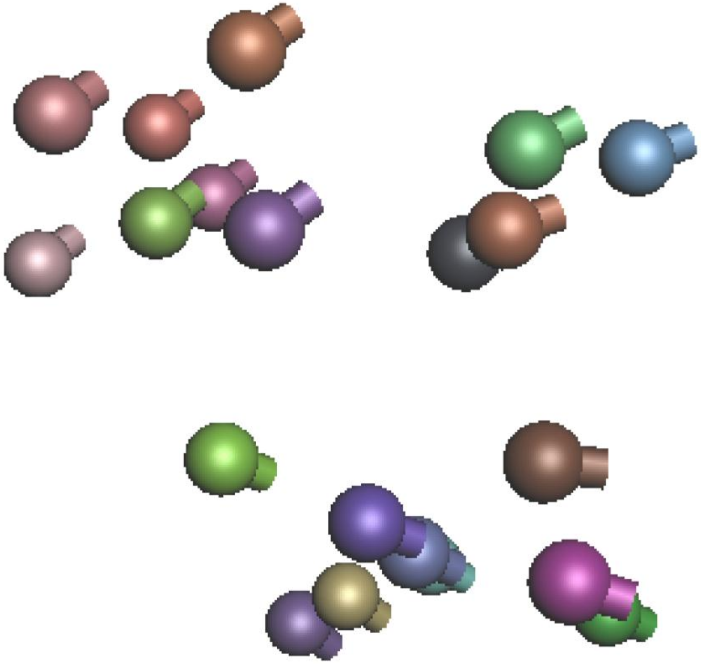
Nvidia GeForce GF100 Graphics Processing Unit Architecture

- **GPC**
  - Graphics Processing Cluster
  - Coherent Level 2 Cache
- **SM**
  - Streaming Multiprocessor
  - In CUDA: Multiprocessor, and the fundamental unit for a thread block
- **Core**
  - Scalar ALU for single CUDA thread
  - IEEE 754-2008 compliant for single and dual precision floating point operations



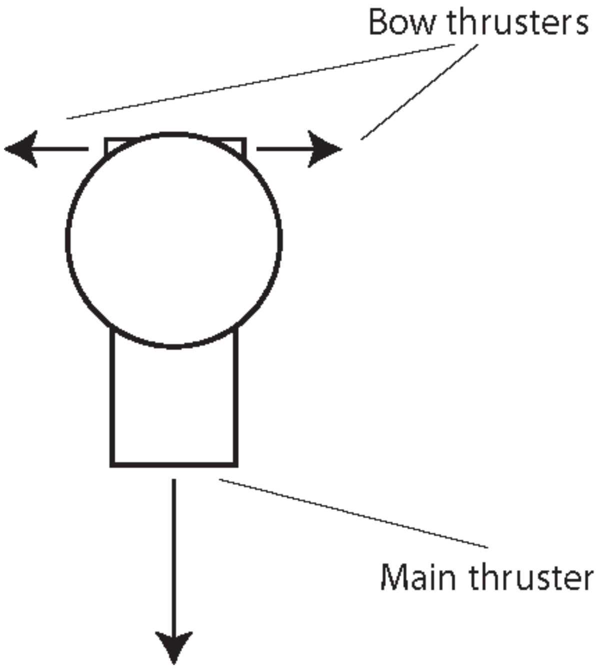
Example Game: The Space Racer

- Simple Space Race Game
- Clients randomly placed in a 3D virtual world.
- Several targets are placed randomly in the 3D virtual world.
- Clients will try to reach the closest target, when a target is reached it will continue to the next.



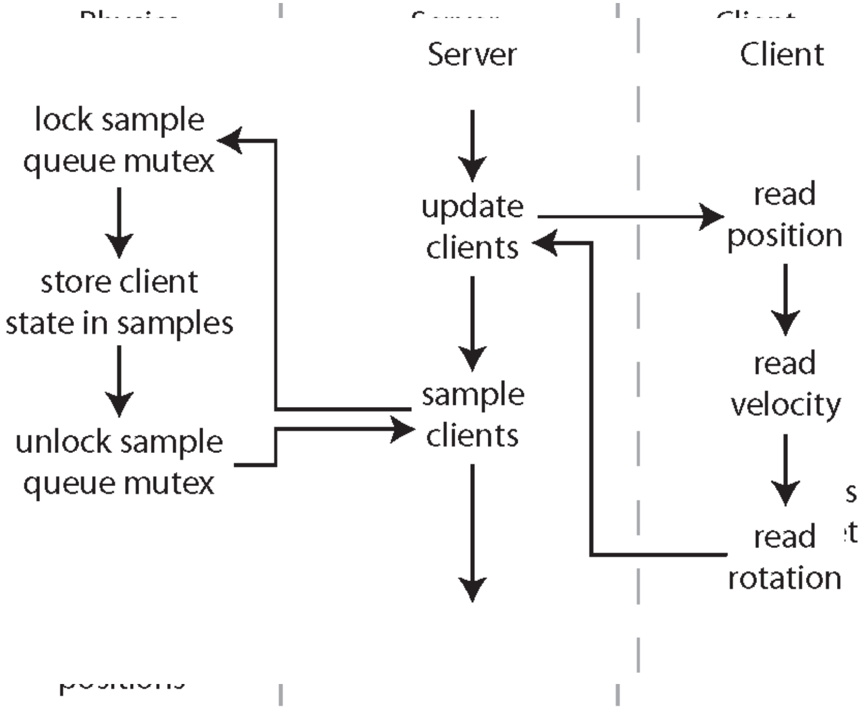
Game Object & Physical Forces in the Game

- Game object have one main thruster for propulsion and two bow thrusters to change course.
- Physical Forces in the game:
  - Linear Motion: Used to implement the vertical gravity, gravity field around the targets, and the main thruster.
  - Angular motion: Used to implement the bow thrusters. This allows the object to rotate in all dimensions



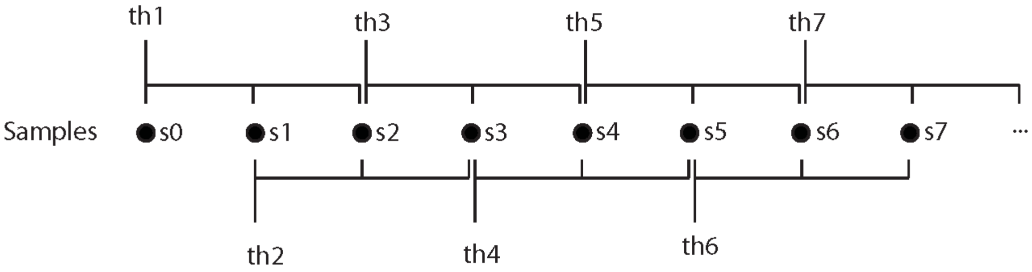
Simulation Structure

- **Generation Mode:**
  - Client will try to reach the closest targets
  - External forces will affect the clients.
  - Write movement data to files, which are used in playback mode.
- **Playback Mode:**
  - Initialize clients and read from files generated in generation mode.
  - Send state data to the server, which will store the samples in queues.



Implementation & Evaluation

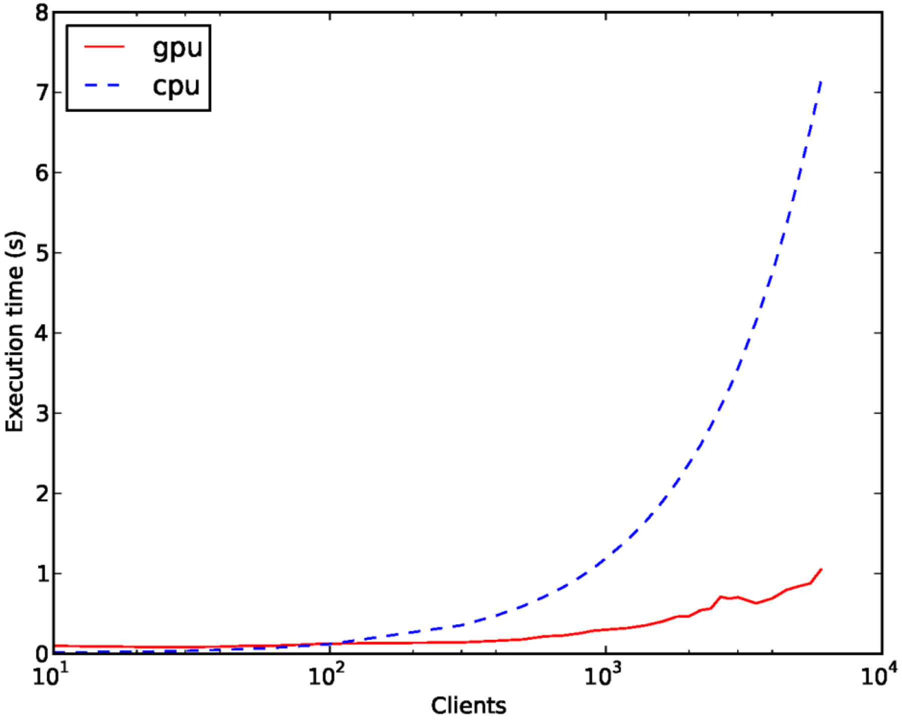
- **Each thread will work on three samples of game state:**
  - A sample contains the movement of a client, and a positional vector
  - The physics engine will be applied in reverse



- **Test Setup:**
  - Intel Core i5 750 2,66 GHz
  - 4 GB RAM
  - Nvidia GeForce GTX 480
  - Nvidia CUDA 3.1

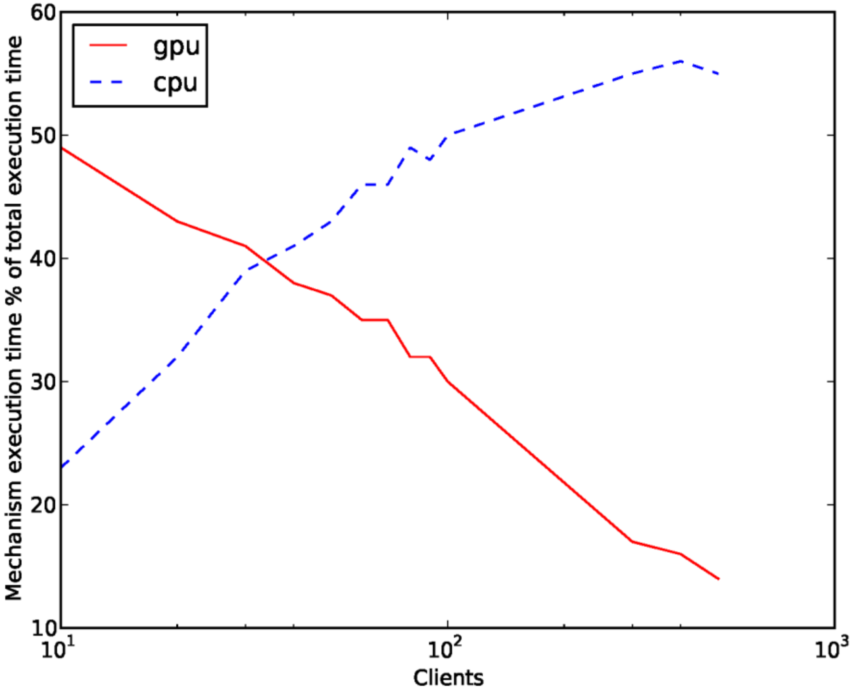
Results: Execution time of the mechanisms

- **Total execution time of the cheat detection system:**
  - Tested 10 to 6000 clients
  - CPU faster with a low number of clients
  - GPU scales good with a higher number of clients.
- Latency for transferring data to the GPU over the PCI Express bus is a challenge for a low numbers of clients.



Results: Percent of time spent on cheat detection

- **Percent of time spent on cheat detection processing:**
  - 10 to 1000 clients
  - With 50 clients, the GPU implementation is spending less time than the CPU version.
- No advanced buffering of samples are currently used. This could improve GPU performance with on a low number of clients



Discussion:

- Cheat detection mechanisms performs differently on GPU and CPU when numbers of clients are increased:
  - Physics operations scale very good on the GPU architecture
  - GPU is more challenging to program compared to a CPU, and the programmer need to think differently.
- Future generations of CPUs from Intel and AMD will integrate a simple GPU on the CPU die, this might reduce the latency of transferring data.
- Further Work:
  - Further optimize the CPU and GPU version of the game.
  - Extend the physical model of the game, and experiment with moving the physics engine to the server.

Conclusion & Further Work

- We have seen that a system processing cheat detection mechanisms on a GPU can outperform the same mechanisms running on a CPU.
- By moving the cheat detection mechanisms to the GPU we are able to offload the CPU allowing the CPU to perform other tasks.

QUESTIONS ?