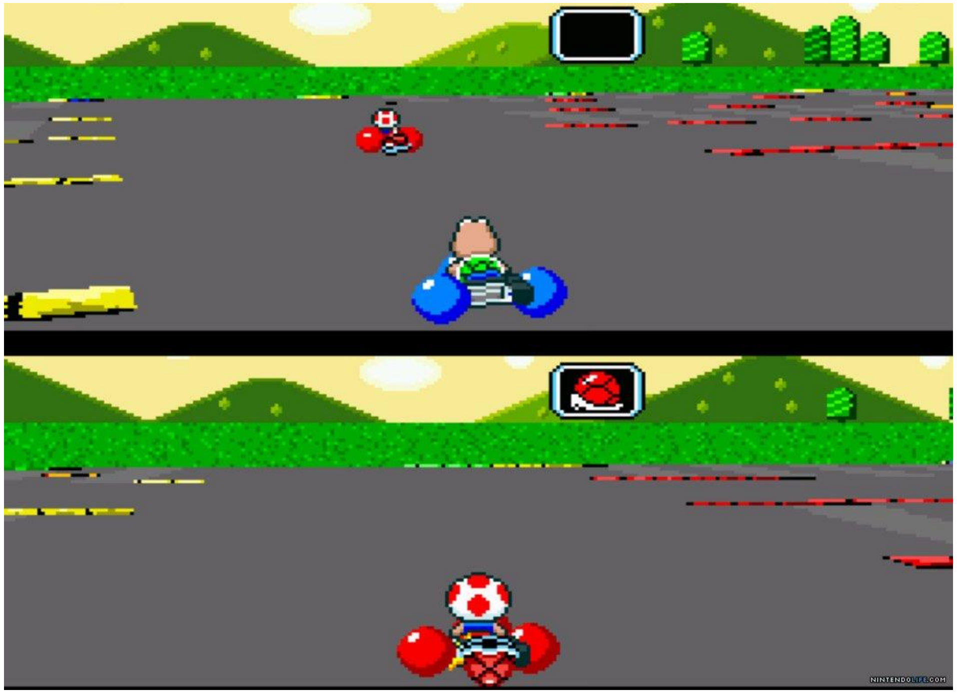
Game Technology

Lecture 12 – 23.01.2015



Dipl-Inf. Robert Konrad
Dr.-Ing. Florian Mehm

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Preliminary timetable

Lecture No.	Date	Topic
1	17.10.2014	Basic Input & Output
2	24.10.2014	Timing & Basic Game Mechanics
3	31.10.2014	Software Rendering 1
4	07.11.2014	Software Rendering 2
5	14.11.2014	Basic Hardware Rendering
6	21.11.2014	Animations
7	28.11.2014	Physically-based Rendering
8	05.12.2014	Physics 1
9	12.12.2014	Physics 2
10	19.12.2014	Procedural Content Generation
11	16.01.2015	Compression & Streaming
12	**23.01.2015**	**Multiplayer**
13	30.01.2015	Audio
14	06.02.2015	Scripting
15	13.02.2015	AI

KONAMI
THE SIMPSONS
KONAMI
The Simpsons
ATTACK
JUMP
ATTACK
JUMP
ATTACK
JUMP
ATTACK
JUMP

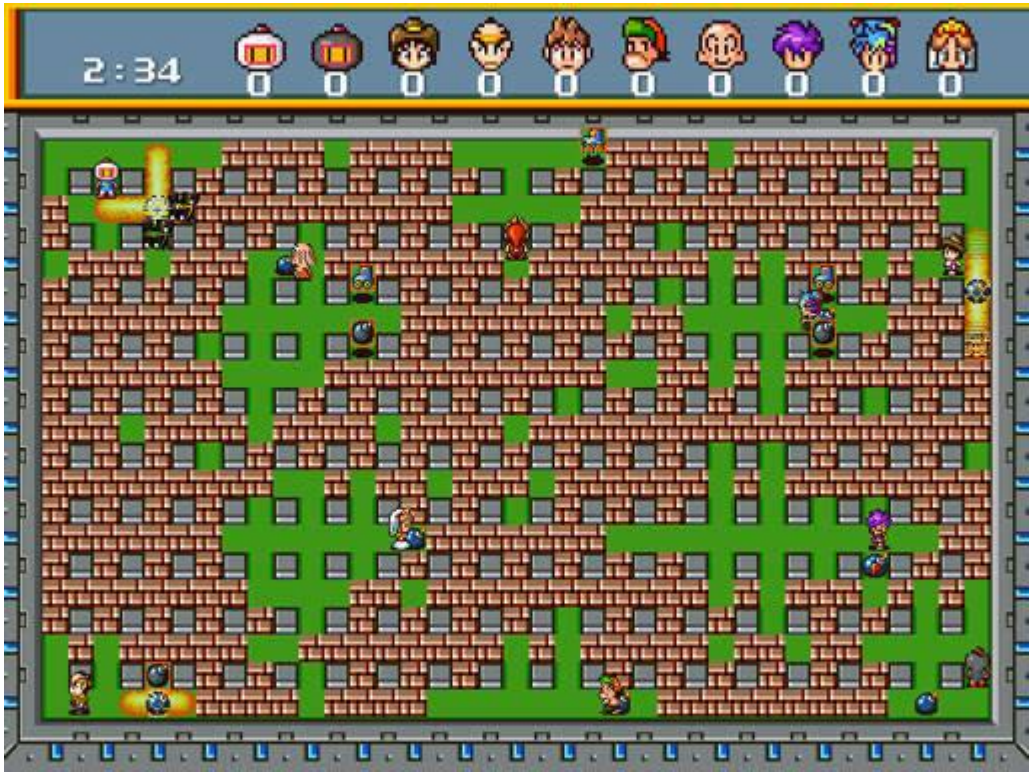
One computer, multiple players

Trivial implementation

No latencies

Uncompressed realtime 3D video chat

Saturn Bomberman



Local multiplayer

Screen space restricted

Number of controllers restricted

Number of locally available players who understand Bomberman severely restricted

www.kewlwallpapers.com

Peer-to-Peer Lockstep

Synchronizes game step by step

- Send command data (go forward, move unit,…)
- Receive commands by all other players
- Simulate game step on all computers
- Repeat

Pro & Contra

Low data rate

- Just high level game commands

Very fragile

- Requires complete determinism
- Requires every client to reliably send data
  - One client hangs -> the game hangs

Maximizes latency

- Game has to wait for every one

Players can‘t join a running game

- Would have to rerun all previous game commands

Determinism

Randomness

- Save your seeds
- Implement your own rand()
- Done

Calculations

- Integer calculations - easy
- Floating point calculations – a little weird
  - Different optimizations on different compilers
    - There is usually a „strict IEEE 754“ option
  - Different CPUs
    - x86 calculates in 80bits, then rounds to 32/64 bit
  - …

Peer-to-Peer Lockstep Today

Still used in strategy games

- Even realtime strategy

Not used in action games

- Because the internetz

Peer-to-Peer Lockstep Today

Game design tricks used to hide latency

- Play an animation/sound immediately
- Move units after all clients agreed

Client/Server

Complete game runs only on the server

- Clients send game commands
- Server sends game state

Server

Simulates the comlete game

- Everything that‘s relevant for the game state
- Including physics
- Not including cosmetics like particle effects

Does not depend on clients

- Clients can hang
- Clients can drop in and out
- Does not result in problems for other clients

Client

Really dump client

- Reads input, sends it to the server
- Does not actually run the game
- Just interpolates received game states
- Might run some simulations for effects work
  - Menu animations
  - Particle effects
  - Physics which do not interfere with gameplay
  - …

Pro & Contra

Very robust

- Clients can hardly cause any problems
- Lags from one client do not propagate to other clients
- No cheating

Very laggy

- Everything lags
  - Even basic movement lags
  - The server simulates every player
- Size of game state has to be rather small

Client/Server today

Outdated

Client/Server with Client-Side Prediction

Mix of Client/Server and a little bit of Peer-to-Peer

Server is still the boss

- But clients predict the game state

Prediction

Prediction

Just run everything on the client and the server

- But no client-client-communication
- Determinism helps

Most of the time, predictions should be correct

- At least for the player character himself
- Makes controls snappy

For other players pure prediction

- Often incorrect

Failed Predictions

Failed Predictions

Use the corrected data

- Cause the server is the boss

Hide your mistakes

- Interpolate visuals to avoid jumps
- Or let stuff jump around when out of view

Failed Predictions

Clients receive only old data

Compare old received data and old predicted data

- When prediction was wrong
  - Recalculate new current state based on received old state
    - Then interpolate

Failed Predictions

Can cause unfair situations

- Visuals show that an enemy was hit but he really wasn‘t

No real solution possible

- Virtual life is not fair :-(

Network Protocols

All IP based

Everything just works like the internet

IP

Internet Protocol

Packet based

- No direct connections
- Much like post packages
- Unreliable

TCP/IP

Direct connections

Reliable streams of data

Super easy

TCP/IP

Builds on a package based protocol

Makes sure every package arrives

Makes sure all packages stay in the same order

TCP/IP

Reorders packages

Requests missing packages again

One missing package can cause huge delays

Missed packages

Unacceptable for most applications

Mostly not important for games

- Positions from 30ms ago are outdated anyway
  - Gets new positions all the time anyway

UDP

Basically IP plus port numbers

Works with packages directly

UDP

Use packages directly for game state

Implement TCP like functionality for other stuff

- Highscore lists,…

UDP

Has additional difficulties

- Applications have to measure transfer rates
- Typical packet sizes (< 512 Bytes) are hopefully enough for one piece of game state

The Future

More Predictions…

Game-Streaming

Run game on the server

Client sends input events

Server sends video stream

Game-Streaming Pro & Contra

Game works like a split-screen game on the server

- Super easy development

Video compression can look ugly

- But internet connections get faster all the time

Latency is as bad or worse than basic Client/Server

Latency

Speed of light is ~300000 km/s

Circumference of the earth ~40000 km

At least one data roundtrip necessary

- > 0.1 seconds for far away servers
  - Too slow

Latency

Streaming Game providers try to place lots of server at different places

- To minimize distance and therefore latency

Typically ends up at speeds that are ok for some persons

- And some genres

Not acceptable for VR

- Super low latency is critical for good VR

Shinra

Research project by Square-Enix

Wants to use streaming to create new types of multiplayer games

Current multiplayer games are restricted by the amount of data that can be transfered

- Doesn’t matter when just streaming audio/video data

Plus want to just use more hardware per game

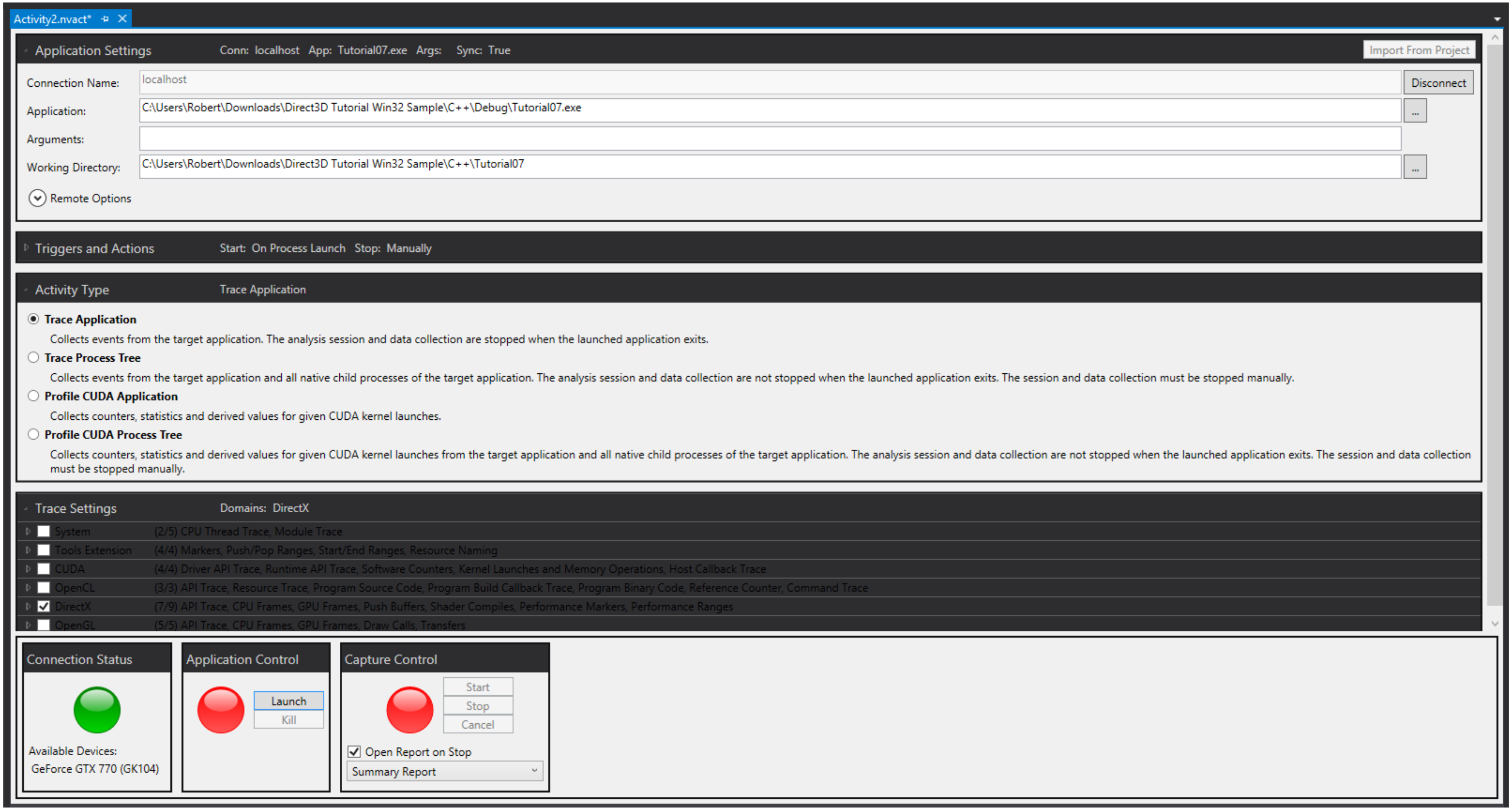
- For more physics or whatever

Alternative strategies?

Maybe cleverly send compressed game state of close surroundings to individual clients

Debugging and Profiling GPU programs

nSight



Application Settings   Conn: localhost App: Tutorial07.exe Args: Sync: True

Connection Name: localhost

Application: C:\Users\Robert\Downloads\Direct3D Tutorial Win32 Sample\C++\Debug\Tutorial07.exe

Arguments:

Working Directory: C:\Users\Robert\Downloads\Direct3D Tutorial Win32 Sample\C++\Tutorial07

Triggers and Actions   Start: On Process Launch Stop: Manually

Activity Type   Trace Application

● **Trace Application**

Collects events from the target application. The analysis session and data collection are stopped when the launched application exits.

○ **Trace Process Tree**

Collects events from the target application and all native child processes of the target application. The analysis session and data collection are not stopped when the launched application exits. The session and data collection must be stopped manually.

○ **Profile CUDA Application**

Collects counters, statistics and derived values for given CUDA kernel launches.

○ **Profile CUDA Process Tree**

Collects counters, statistics and derived values for given CUDA kernel launches from the target application and all native child processes of the target application. The analysis session and data collection are not stopped when the launched application exits. The session and data collection must be stopped manually.

Trace Settings   Domains: DirectX

□ System	(2/5) CPU Thread Trace, Module Trace
□ Tools Extension	(4/4) Markers, Push/Pop Ranges, Start/End Ranges, Resource Naming
□ CUDA	(4/4) Driver API Trace, Runtime API Trace, Software Counters, Kernel Launches and Memory Operations, Host Callback Trace
□ OpenCL	(3/3) API Trace, Resource Trace, Program Source Code, Program Build Callback Trace, Program Binary Code, Reference Counter, Command Trace
☑ DirectX	(7/9) API Trace, CPU Frames, GPU Frames, Push Buffers, Shader Compiles, Performance Markers, Performance Ranges
□ OpenGL	(5/5) API Trace, CPU Frames, GPU Frames, Draw Calls, Transfers

Connection Status

Available Devices:
GeForce GTX 770 (GK104)

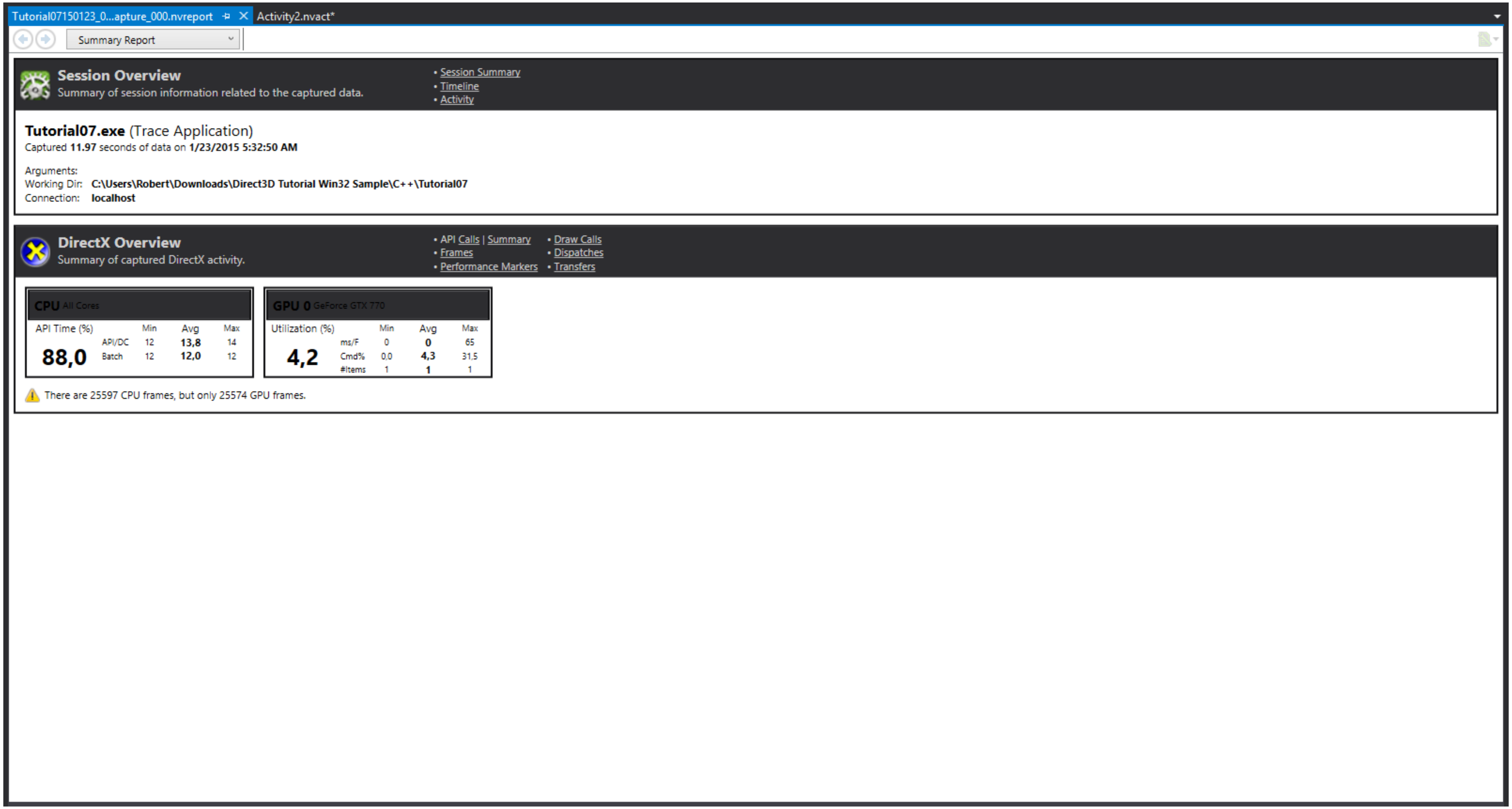
Application Control

Capture Control

☑ Open Report on Stop

Summary Report

nSight



Session Overview

Summary of session information related to the captured data.

- Session Summary
- Timeline
- Activity

Tutorial07.exe (Trace Application)

Captured **11.97** seconds of data on **1/23/2015 5:32:50 AM**

Arguments:
Working Dir: **C:\Users\Robert\Downloads\Direct3D Tutorial Win32 Sample\C++\Tutorial07**
Connection: **localhost**

DirectX Overview

Summary of captured DirectX activity.

- API Calls | Summary
- Frames
- Performance Markers
- Draw Calls
- Dispatches
- Transfers

CPU All Cores

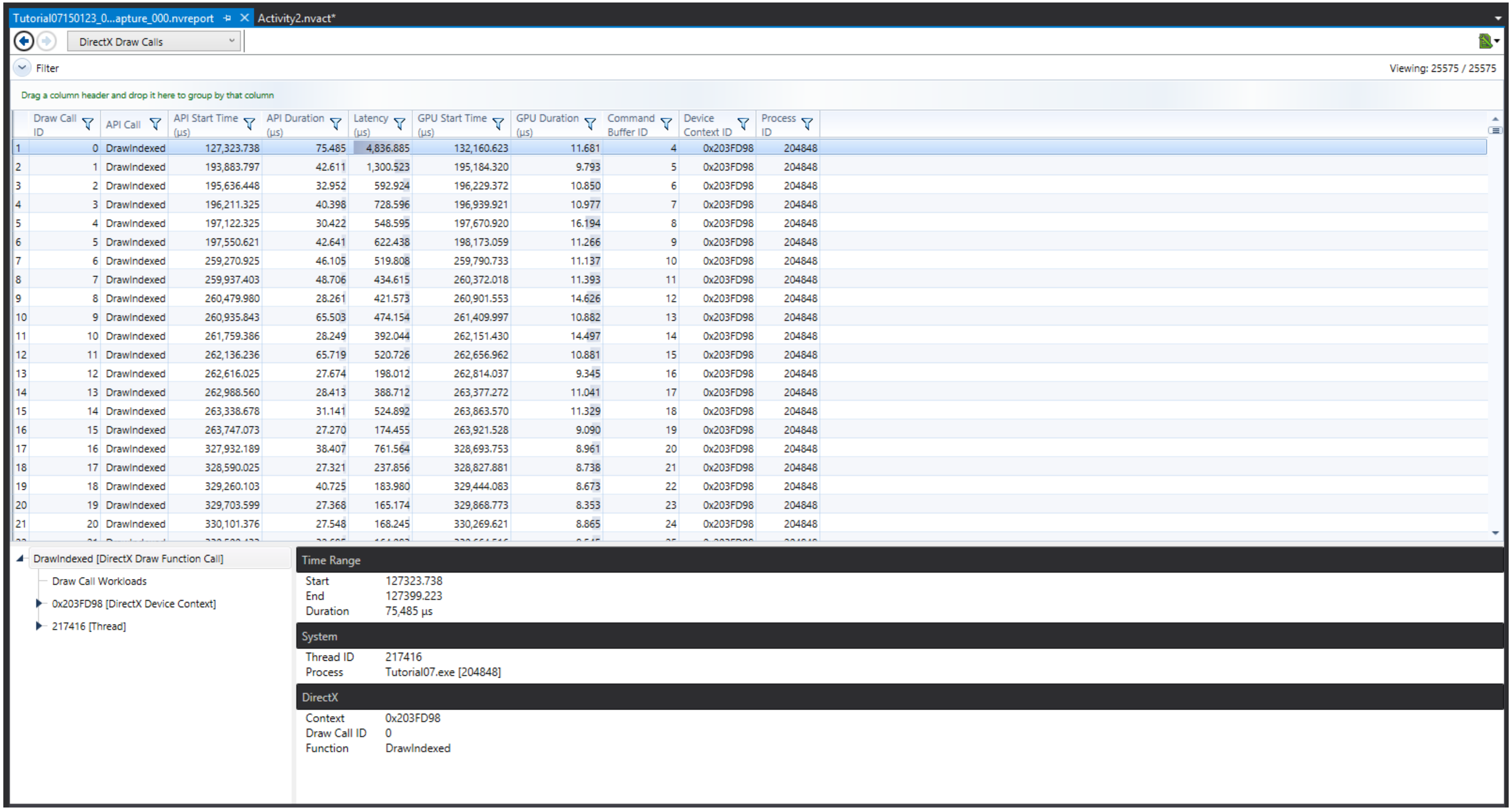
API Time (%)		Min	Avg	Max
88,0	API/DC	12	**13,8**	14
	Batch	12	**12,0**	12

GPU 0 GeForce GTX 770

Utilization (%)		Min	Avg	Max
4,2	ms/F	0	**0**	65
	Cmd%	0,0	**4,3**	31,5
	#Items	1	**1**	1

⚠ There are 25597 CPU frames, but only 25574 GPU frames.

nSight



Tutorial07150123_0...apture_000.nvreport | Activity2.nvact*

DirectX Draw Calls

Filter

Viewing: 25575 / 25575

Drag a column header and drop it here to group by that column

	Draw Call ID	API Call	API Start Time (µs)	API Duration (µs)	Latency (µs)	GPU Start Time (µs)	GPU Duration (µs)	Command Buffer ID	Device Context ID	Process ID
1	0	DrawIndexed	127,323.738	75.485	4,836.885	132,160.623	11.681	4	0x203FD98	204848
2	1	DrawIndexed	193,883.797	42.611	1,300.523	195,184.320	9.793	5	0x203FD98	204848
3	2	DrawIndexed	195,636.448	32.952	592.924	196,229.372	10.850	6	0x203FD98	204848
4	3	DrawIndexed	196,211.325	40.398	728.596	196,939.921	10.977	7	0x203FD98	204848
5	4	DrawIndexed	197,122.325	30.422	548.595	197,670.920	16.194	8	0x203FD98	204848
6	5	DrawIndexed	197,550.621	42.641	622.438	198,173.059	11.266	9	0x203FD98	204848
7	6	DrawIndexed	259,270.925	46.105	519.808	259,790.733	11.137	10	0x203FD98	204848
8	7	DrawIndexed	259,937.403	48.706	434.615	260,372.018	11.393	11	0x203FD98	204848
9	8	DrawIndexed	260,479.980	28.261	421.573	260,901.553	14.626	12	0x203FD98	204848
10	9	DrawIndexed	260,935.843	65.503	474.154	261,409.997	10.882	13	0x203FD98	204848
11	10	DrawIndexed	261,759.386	28.249	392.044	262,151.430	14.497	14	0x203FD98	204848
12	11	DrawIndexed	262,136.236	65.719	520.726	262,656.962	10.881	15	0x203FD98	204848
13	12	DrawIndexed	262,616.025	27.674	198.012	262,814.037	9.345	16	0x203FD98	204848
14	13	DrawIndexed	262,988.560	28.413	388.712	263,377.272	11.041	17	0x203FD98	204848
15	14	DrawIndexed	263,338.678	31.141	524.892	263,863.570	11.329	18	0x203FD98	204848
16	15	DrawIndexed	263,747.073	27.270	174.455	263,921.528	9.090	19	0x203FD98	204848
17	16	DrawIndexed	327,932.189	38.407	761.564	328,693.753	8.961	20	0x203FD98	204848
18	17	DrawIndexed	328,590.025	27.321	237.856	328,827.881	8.738	21	0x203FD98	204848
19	18	DrawIndexed	329,260.103	40.725	183.980	329,444.083	8.673	22	0x203FD98	204848
20	19	DrawIndexed	329,703.599	27.368	165.174	329,868.773	8.353	23	0x203FD98	204848
21	20	DrawIndexed	330,101.376	27.548	168.245	330,269.621	8.865	24	0x203FD98	204848

- DrawIndexed [DirectX Draw Function Call]
  - Draw Call Workloads
  - 0x203FD98 [DirectX Device Context]
  - 217416 [Thread]

Time Range

Start	127323.738
End	127399.223
Duration	75,485 µs

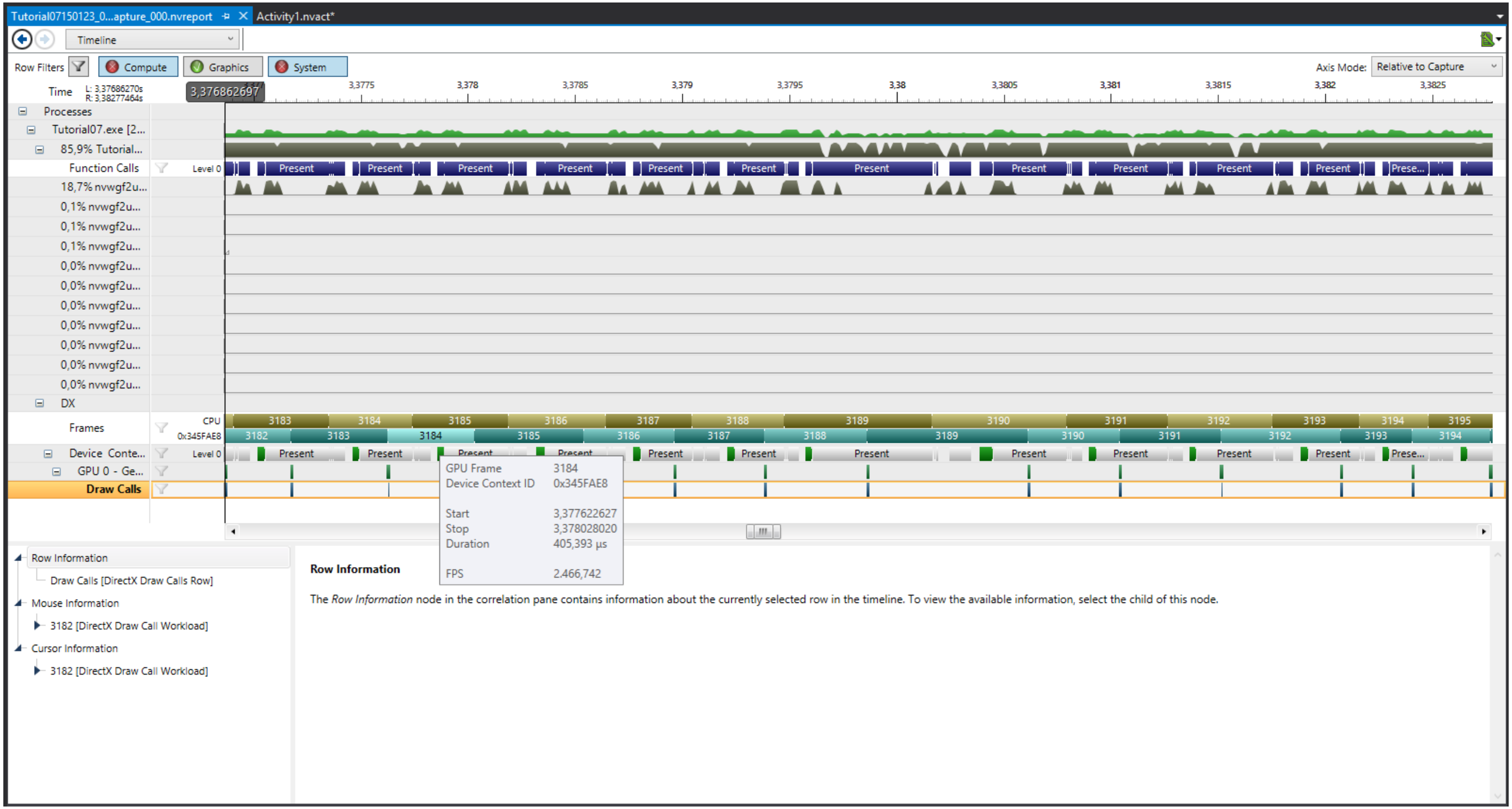
System

Thread ID	217416
Process	Tutorial07.exe [204848]

DirectX

Context	0x203FD98
Draw Call ID	0
Function	DrawIndexed

nSight



GPU Frame 3184
Device Context ID 0x345FAE8

Start 3,377622627
Stop 3,378028020
Duration 405,393 µs

FPS 2.466,742

Row Information

The *Row Information* node in the correlation pane contains information about the currently selected row in the timeline. To view the available information, select the child of this node.

Direct3D11 and OpenGL now have performance query apis

- ID3D11Query
- ARB_timer_query

2.1 Hardware

What makes it so important that texture compression algorithms are directly supported by the hardware?

Reading pixels is the most fundamental and speed critical operation of GPUs.

2.2 Artifacts

ETC is a lossy texture compression algorithm. Describe what characteristics an image should have to make those losses clearly visible.

Big contrasts across block borders.

2.3 Tilemaps

Outline an algorithm to display tilemaps correctly in a 3D environment.

MegaTextures for example.