Mobile Graphics
Siggraph Asia 2017 course

Marco Agus, KAUST & CRS4
Enrico Gobbetti, CRS4
Fabio Marton, CRS4
Giovanni Pintore, CRS4
Pere-Pau Vázquez, UPC

November 2017
WELCOME TO THIS HALF-DAY COURSE!
Subject: Mobile Graphics

- All you need to know to get an introduction to the field of mobile graphics:
  - Scope and definition of “mobile graphics”
  - Brief overview of current trends in terms of available hardware architectures and research apps built of top of them
  - Quick overview of development environments
  - Rendering, with focus on rendering massive/complex surface and volume models
  - Capture, with focus on data fusion techniques
Speakers (in alphabetical order)

- Marco Agus (1,2)
  - Research Engineer at KAUST (Saudi Arabia)
  - Researcher at CRS4 (Italy)

- Enrico Gobbetti (1) - Organizer
  - Director of Visual Computing at CRS4 (Italy)

- Fabio Marton (1)
  - Researcher at CRS4

- Giovanni Pintore (1)
  - Researcher at CRS4

- Pere-Pau Vázquez (3)
  - Professor at UPC, Spain

(1) www.crs4.it/vic/
(2) https://vcc.kaust.edu.sa
(3) http://www.virvig.eu/
Funding

Center for Research, Development, and Advanced Studies in Sardinia, Italy

King Abdullah University of Science & Technology, Saudi Arabia

Polytechnic University of Catalonia, Spain

Project TDM
RAS - POR FESR 2014-2020

REGIONE AUTÓNOMA DE SARDIGNA
REGIONE AUTONOMA DELLA SARDEGNA
Projects VIGEC / VIDEOLAB

Spanish MINECO Ministry FEDER funds
Grant No. TIN2014-52211-C2-1-R
## Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Presenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>5’</td>
<td>0. Introduction and outline</td>
<td>Enrico</td>
</tr>
<tr>
<td>15’</td>
<td>1. Evolution of Mobile Graphics</td>
<td>Marco</td>
</tr>
<tr>
<td>20’</td>
<td>2.1 Mobile Graphics Trends / Hardware</td>
<td>Pere-Pau</td>
</tr>
<tr>
<td>15’</td>
<td>2.2 Mobile Graphics Trends / Applications</td>
<td>Marco</td>
</tr>
<tr>
<td>5’</td>
<td>4.1 Scalable Mobile Visualization / Introduction</td>
<td>Enrico</td>
</tr>
<tr>
<td>30’</td>
<td>4.2 Scalable Mobile Visualization / Massive Meshes</td>
<td>Fabio</td>
</tr>
<tr>
<td>15’</td>
<td>BREAK</td>
<td>-</td>
</tr>
<tr>
<td>5’</td>
<td>4.3 Scalable Mobile Visualization / Intro to complex lighting</td>
<td>Enrico</td>
</tr>
<tr>
<td>10’</td>
<td>4.4 Scalable Mobile Visualization / Lighting Precomputation</td>
<td>Fabio</td>
</tr>
<tr>
<td>20’</td>
<td>4.5 Scalable Mobile Visualization / Smart Shading</td>
<td>Pere-Pau</td>
</tr>
<tr>
<td>15’</td>
<td>4.6 Scalable Mobile Visualization / Volumes</td>
<td>Pere-Pau</td>
</tr>
<tr>
<td>10’</td>
<td>5.1 Mobile Metric Capture and Reconstruction / Introduction</td>
<td>Enrico</td>
</tr>
<tr>
<td>30’</td>
<td>5.2 Mobile Metric Capture and Reconstruction / Case studies</td>
<td>Gianni</td>
</tr>
<tr>
<td>15’</td>
<td>6. Closing and Q&amp;A</td>
<td>ALL</td>
</tr>
</tbody>
</table>
Next Session

EVOLUTION OF MOBILE GRAPHICS
Part 1

Evolution of the mobile graphics world

Marco Agus, KAUST & CRS4
Mobile evolution (1/3)

1980s
Talk time of 30 minutes and 10 hours to re-charge.

1990s
Bigger screens and smaller smarter phones.
The first clamshell and with vibrate function.
The term "smart-phone" was first pronounced in 1997.
First colour screen phone, displaying red, green, blue and white.
In 1998 sold more devices than cars and PCs combined.
First mobile phone featuring WAP.

Infographic designed by LEWIS PR for Mobile World Barcelona
Mobile evolution (1/3)

**1980s**
- Talk time of 30 minutes and 10 hours to re-charge.
- Bigger screens and smaller smarter phones.

**1990s**
- The first clamshell and with vibrate function.
- The term "smart-phone" was first pronounced in 1997.
- First colour screen phone, displaying red, green, blue and white.
- In 1998 sold more devices than cars and PCs combined.
- First mobile phone featuring WAP.

Color display

Infographic designed by LEWIS PR for Mobile World Barcelona
Mobile evolution (2/3)

2000s

- The first mobile phone with T9 Predictive text.
- The first cellphone to feature an internal camera.
- Referred to as the "lipstick" phone.
- 1.9 trillion text messages were sent worldwide in 2007.
- In 2007, possible most feature rich phone to date.
- First phone shipped with an OS as advanced as the desktop version.

Infographic designed by LEWIS PR for Mobile World Barcelona
Mobile evolution (2/3)

Smartphones, OS

2000s

- The first mobile phone with T9 Predictive text.
- The first cellphone to feature an internal camera.
- Referred to as the "lipstick" phone.
- 1.9 trillion text messages were sent worldwide in 2007.
- In 2007, possible most feature rich phone to date.
- First phone shipped with an OS as advanced as the desktop version.

Infographic designed by LEWIS PR for Mobile World Barcelona
Mobile evolution (3/3)

- **2010s**
  - **Galaxy Samsung**: Samsung developed a phone with "Smart Stay" and "Smart Alert" and S Voice.
  - **Xperia Z**: In 2013 Sony announced the world's first waterproof phone.
  - **Galaxy Note**: In January 2013 Samsung launched their flexible OLED displays, calling them YOUM.
  - **Yota**: It is predicted that the smartphone or tablet of the future will be as flexible as paper.

Infographic designed by LEWIS PR for Mobile World Barcelona
High resolution displays

2010s

Samsung developed a phone with “Smart Stay” and “Smart Alert” and S Voice.

In 2013 Sony announced the world’s first waterproof phone.

In January 2013 Samsung launched their flexible OLED displays, calling them YOUm.

It is predicted that the smartphone or tablet of the future will be as flexible as paper.

Infographic designed by LEWIS PR for Mobile World Barcelona
Mobile evolution (3/3)

Similar evolution for PDAs, and tablets

- **2010s**
  - Samsung developed a phone with “Smart Stay” and “Smart Alert” and S Voice.
  - In 2013 Sony announced the world’s first waterproof phone.
  - In January 2013 Samsung launched their flexible OLED displays, calling them YOUM.
  - It is predicted that the smartphone or tablet of the future will be as flexible as paper.

Infographic designed by LEWIS PR for Mobile World Barcelona
Mobile connectivity evolution

- Bandwidth is doubling every 18 months
- Mobile internet users overcame desktop internet users
- 2017 smartphone traffic expected at 2.7 GB per person per month

© www.statista.com
Displays and User Interface

• Before 2007 – old days
  – PDA → Palm OS/ Windows Pocket / Windows CE
  – Stylus interaction (touch screens at early stages)

• Touch era
  – 2007 – iOS /iPhone
  – 2008 – Android / HTC Dream or G1
  – Touch-enabled devices (no stylus required)

• Nowadays
  – Wearables → <2”
  – Smartphones → 3-6”
  – Tablets → >7-10”
  – DLP projectors integrated
## Display characteristics

<table>
<thead>
<tr>
<th>Application</th>
<th>100 PPI</th>
<th>150 PPI</th>
<th>200 PPI</th>
<th>250 PPI</th>
<th>300 PPI</th>
<th>400 PPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 cm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smart Phone</td>
<td>3.5&quot; 400 × 234 (132 PPI)</td>
<td>3.5&quot; 480 × 320 (164 PPI)</td>
<td>3.5&quot; 800 × 480 (266 PPI)</td>
<td>3.5&quot; 960 × 640 (326 PPI)</td>
<td>3.5&quot;/3.7&quot; 1280 × 800 (400 PPI)</td>
<td>500 ppi</td>
</tr>
<tr>
<td>Tablet PC</td>
<td>7&quot; 800 × 480 (133 PPI)</td>
<td>7&quot; 1024 × 600 (169 PPI)</td>
<td>7&quot; 1280 × 800 (215 PPI)</td>
<td>9.7&quot; 1600 × 1200 (206 PPI)</td>
<td>9.7&quot; 2048 × 1536 (264 PPI)</td>
<td>5.1&quot;/5.5&quot; 2560x1440 (&gt;500 PPI)</td>
</tr>
<tr>
<td></td>
<td>9.7&quot; 1024 × 768 (132 PPI)</td>
<td>9.7&quot; 1600 × 1200 (206 PPI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Note</td>
<td>10.1&quot; 1024 × 600 (118 PPI)</td>
<td>10.1&quot; 1280 × 800 (150 PPI)</td>
<td>10.1&quot; 1920 × 1080 (210 PPI)</td>
<td>10.1&quot; 2560 × 1600 (300 PPI)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notebook PC</td>
<td>15.6&quot; 1366 × 768 (110 PPI)</td>
<td>14.0&quot; 1366 × 768 (110 PPI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCD MNT</td>
<td>21.5&quot; 1920 × 1080 (100 PPI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

50 cm          |              |              |              |              |              |              |

60 cm          |              |              |              |              |              |              |
Chip evolution (1/2)

Scalable Mobile Processor Evolution

Relative Performance


Cortex-A8 Mali-200
Cortex-A8 Mali-300
Cortex-A9 Quad Mali-400 MP
Quad Cortex-A8 Quad Mali-T604
Dual Cortex-A9 Quad Mali-T604
Dual Cortex-A15 Dual Cortex-A7 Quad Mali-T658
Cortex-A15 Cortex-A7 Dual Mali-T604
Dual Cortex-A15 Dual Cortex-A7 Eight Mali-T658

Superphone
Mid Range
Entry Level

© ARM
Chip evolution (2/2)

The killer mobile processors™

- Microprocessors killed the Vector supercomputers
  - They were not faster ...
  - ... but they were significantly cheaper and greener

- History may be about to repeat itself ...
  - Mobile processor are not faster ...
  - ... but they are significantly cheaper

Scenario

• Modern smartphones (tablets) are compact visual computing powerhouses
• DIFFUSION: more than 4.6 billion mobile phone subscriptions
  – [Ellison 2010]
• NETWORKING: High speed internet connection (typical 1GB/month plan)
  – 3G - < 0.6-3Mbps ~ 100KB/s - 400KB/s (latency ~ 100-125ms)
  – 4G – < 3-10Mbps ~ 400KB/s - 1MB/s (latency ~ 60-70ms)
  – 5G - 1Gbps (from 2016?)
• MEMORY: Increasing RAM and storage space
  – RAM 1-3GB
  – Storage 8-64GB
• COMPUTING: Increasing processing power
  – CPU 4-8 core @ 2.5Ghz
  – GPU 72-192 cores (~ALUs)
Scenario

• More than 4.6 billion mobile phone subscriptions
  – [Ellison 2010]

• High speed internet connection (typical 1GB/month plan)
  – 3G – < 0.6-3Mbps ~ 100KB/s - 400KB/s
  – 4G – < 3-10Mbps ~ 400KB/s - 1MB/s

• Increasing RAM and storage space
  – RAM 1-3GB
  – Storage 8-64GB

• Increasing processing power
  – CPU 4-8 core @ 2.5Ghz
  – GPU 72-192 cores (~ALUs)
Where are we going?

- Powerful devices for acquiring, processing and visualizing information
- Accessibility of information (anybody, any time, anywhere)
- Immense potential (integration of acquisition, processing, visualization, cloud computing, and collaborative tasks)
Next Session

MOBILE GRAPHICS TRENDS: HARDWARE ARCHITECTURES & APPLICATIONS
Part 2.1

Mobile Graphics Trends: Hardware Architectures

Pere-Pau Vázquez, UPC
Architectures (beginning 2015)
Architectures

• x86 (CISC 32/64bit)
  – Intel Atom Z3740/Z3770, X3/X5/X7
  – AMD Amur / Styx (announced)
  – Present in few smartphones, more common in tablets
  – Less efficient

• ARM
  – RISC 32/64bit
    • With SIMD add-ons
  – Most common chip for smartphones
  – More efficient & smaller area

• MIPS
  – RISC 32/64bit
  – Including some SIMD instructions
  – Acquired by Imagination, Inc. @2014
Architectures – RISC vs. CISC but...

- **CISC (Complex Instruction Set Computer)**
  - Fast program execution (optimized complex paths)
  - Complex instructions (i.e. memory-to-memory instructions)

- **RISC (Reduced Instruction Set Computer)**
  - Fast instructions (fixed cycles per instruction)
  - Simple instructions (fixed/reduced cost per instruction)

- **FISC (Fast Instruction Set Computer)**
  - Current RISC processors integrate many improvements from CISC: superscalar, branch prediction, SIMD, out-of-order
  - Philosophy → fixed/reduced cycle count/instr
  - Discussion (Post-RISC):
Landscape has changed a bit...

• **Status by 2014-2015:**
  - Intel Atom X3/X5/X7 announced (March 2015)
  - AMD announces Amur / Styx (20nm, Oct. 2014)
  - Nvidia launches Tegra X1 (March 2015)
  - ARM the only EU big technology company
  - Imagination announces Furian (sub 14nm, March 2017) Imagination’s chips are in iPhones & iPads

• **Nowadays:**
  - Intel quits mobile Apr/May 2016
  - AMD cancels 20nm chips (Jul. 2015)
  - Nvidia cancels Shield tablet (Aug. 2016)
  - ARM acquired by Softbank (Sep. 2016)
  - Apple tells Imagination that their IP will not be needed in 18-24 months (Apr. 2017)
Architectures (nowadays)

Acquired by Imagination, inc
Architectures – ARM

- ARM Ltd.
  - RISC processor (32/64 bit)
  - IP (intellectual property) – Instruction Set / ref. implementation
  - CPU / GPU (Mali)
- Licenses (instruction set OR ref. design)
  - Instruction Set license -> custom made design (SnapDragon, Samsung in Galaxys, Apple in iPhones & iPads)
    - Optimizations (particular paths, improved core freq. control,...)
  - Reference design (Cortex A9, Cortex A15, Cortex A53/A57...)
- Licensees (instruction set OR ref. design)
  - Apple, Qualcomm, Samsung, Nvidia, AMD, MediaTek, Amazon (through Annapurna Labs, Inc.)...
  - Few IS licenses, mostly adopting reference design
- Manufacturers
  - Contracted by Licensees
    - GlobalFoundries, United Microelectronics, TSM...
Architectures – ARM...

• Supported on
  – Android, iOS, Win Phone, Tizen, Firefox OS, BlackBerry, Ubuntu Phone, …

• Biggest mobile market share

• Typically paired with mobile GPUs. Existing brands:
  – Adreno 4x0/5x0 – Qualcomm
  – PowerVR 8XE (Rogue) – Imagination
  – Mali T8x0/G51/G71 – ARM

• General strategies:
  – Cache coherence – week sequential code guarantees on multithreading!!
  – Heavy dependence on compiler → optimize instruction scheduling
    • Operation dependencies, loop unrolling, etc…
  – Use SIMD extensions
Architecture types

- **High performance**
  - Premium smartphones & tablets

- **High area efficiency**
  - Medium-to-low smartphones

- **Ultra-low power**
  - Smartwatches
Architectures

Mobile GPU architecture trends
Graphics pipeline trends

- Tiled rendering
- Data (texture) compression
- Other optimizations
Tiled Rendering

- Immediate Mode Rendering (IMR)
- Tile-Based Rendering (TBR)
- Tile-Based Deferred Rendering (TBDR)
Architectures – GPU

- **Immediate Mode Rendering (IMR)**
  - Geometry is processed in submission order
    - High **overdraw** (shaded pixels can be overwritten)
  - Buffers are kept in System Memory
    - High bandwidth / power / latency
  - Early-Z helps depending on geometry sorting
    - Depth buffer value closer than fragment → discard

Architectures – GPU

- **Tile Based Rendering (TBR)**
  - Rasterizing per-tile (triangles in bins per tile) 16x16, 32x32
    - Buffers are kept on-chip memory (GPU) – fast! ⇒ *geometry limit?*
  - Triangles processed in submission order (TB-IMR)
    - Overdraw (front-to-back -> early z cull)
  - Early-Z helps depending on geometry sorting

---

Architectures – GPU

**Tile Based Deferred Rendering (TBDR)**
- Fragment processing (tex + shade) ~waits for Hidden Surface Removal
  - Micro Depth Buffer – depth test before fragment submission
    - whole tile → 1 frag/pixel 😊
  - iPad 2X slower than Desktop GeForce at HSR (FastMobileShaders_siggraph2011)
- Possible to prefetch textures before shading/texturing
- Hard to profile!!! ~~~Timing?

Data/texture compression

- ARM’s Adaptive Scalable Texture Compression (ASTC) supported by most mobile GPU vendors
- ETC2/EAC standard compression OpenGL ES 3.0
- Compression hardware also present in display hardware
  - Rendered images stored and transferred to the display in a compressed
    - Saving bandwidth
Other optimizations

• Deferred shading
• Primitive elimination
• Skipping updates to pixels that do not change
  – ARM memory transaction elimination
Trends

• Specific hardware for ray tracing
• Learning libraries & hardware (e.g. Qualcomm’s Fast CV, Nvidia’s CUDA Deep Neural Network)
• Skipping updates to pixels that do not change
  – ARM memory transaction elimination
Part 2.2

Mobile Graphics Trends: Applications

Marco Agus, KAUST & CRS4
Applications

• Wide range of applications
  – Cultural Heritage
  – Medical Image
  – 3D object registration
  – GIS
  – Gaming
  – VR & AR
  – Building reconstruction
  – Virtual HCI
Mobile 3D interactive graphics

- General pipeline similar to standard interactive applications

![Diagram of Mobile 3D interactive graphics]

1. DATA ACCESS
2. Scene
3. RENDERING
4. Frame
5. DISPLAY
6. INTERACTION
Remote rendering

- General solution since first PDAs
Remote rendering

- 3D graphics applications require intensive computation and network bandwidth
  - electronic games
  - visualization of very complex 3D scenes
- Remote rendering has long history and it is successfully applied for gaming services
  - Limitation: interaction latency in cellular networks
Mixed Mobile/Remote rendering

- As mobile GPUs progress...
Mixed Mobile/Remote rendering

- Model based versus Image based methods
- Model based methods
  - Original models
  - Partial models
  - Simplified models
    - Couple of lines
    - Point clouds

Eisert and Fechteler. Low delay streaming of computer graphics (ICIP 2008)


Balsa et al., Compression-domain Seamless Multiresolution Visualization of Gigantic Meshes on Mobile Devices (Web3D 2013)


Mixed Mobile/Remote rendering

• Model based versus Image based methods
• Model based methods

Point clouds organized as hierarchical grids. Tested on PDAs

• Point clouds

Mixed Mobile/Remote rendering

- Model based versus Image based methods
- Model based methods

Transfer couple of 2D line primitives over the network, which are rendered locally by the mobile device

- Couple of lines
- Point clouds


Mixed Mobile/Remote rendering

• Model based versus Image based methods
• Model based methods
  – Original models

Eisert and Fechteler. *Low delay streaming of computer graphics* (ICIP 2008)

Intercept and stream OpenGL commands
Better performances with respect to video streaming
Limitation: clients need powerful GPU
Mixed Mobile/Remote rendering

- Model based versus Image based methods
- Model based methods
  - Original models
  - Partial models
  - Simplified models
    - Couple of lines
    - Point clouds

More details in Part 4

- Eisert and Fechteler. *Low delay streaming of computer graphics* (ICIP 2008)
- Balsa et al., *Compression-domain Seamless Multiresolution Visualization of Gigantic Meshes on Mobile Devices* (Web3D 2013)
## Mixed Mobile/Remote rendering

- **Image based methods**
  - **Image impostors**
  - **Environment maps**
  - **Depth images**
Mixed Mobile/Remote rendering

- **Image based methods**
  - **Image impostors**
  - **Environment maps**
  - **Depth images**

*Image representations are created by the server, and warped in real time by the client to account for user interaction.*
Mixed Mobile/Remote rendering

- **Model based vs Image based methods**
  - Constraints: rendering quality, bandwidth, interactivity

  - Model based
  - Image based

  - Partial models
    - Original models
    - Simplified models

  - Depth images
    - Environment maps
    - Image impostors

  - Network bandwidth
  - Client computation
  - Rendering quality
  - Occlusion warping limitations
Mobile visualization systems

- **Volume rendering**
  


- **Point cloud rendering**
  
  Balsa et al. *Interactive exploration of gigantic point clouds on mobile devices*. (VAST 2012)

  He et al. *A multiresolution object space point-based rendering approach for mobile devices* (AFRIGRAPH, 2007)
Mobile visualization systems

• **Volume rendering**


• **Point cloud rendering**

  Balsa et al. *Interactive exploration of gigantic point clouds on mobile devices*. (VAST 2012)

  He et al. *A multiresolution object space point-based rendering approach for mobile devices* (AFRIGRAPH, 2007)

*See section 4 for details*
Mobile rendering

• Nowadays...

MOBILE DEVICE

DATA ACCESS → RENDERING → DISPLAY

Scene → Frame

INTERACTION
Mobile rendering

• Or better...

[Diagram showing mobile rendering process with nodes labeled: Server, Data Access, Scene, Rendering, Frame, Display, Interaction, Network]
Mobile rendering

- Or better...

Chunk-based data streaming (like HuMoRS Balsa et al. 2014)

Limitations: bandwidth consumption (for now)
Mobile rendering with capture

- Exploiting mobile device sensors...

Diagram:
- MOBILE DEVICE
  - Environment
  - Scene
  - Frame

Data Access
- Capture
- Rendering
- Display
- Interaction
Mobile rendering with capture

- Exploiting mobile device sensors...

Kolev et al. *Turning Mobile Phones into 3D Scanners* (CVPR 2014)

Tanskanen et al. *Live Metric 3D Reconstruction on Mobile Phones* (ICCV 2013)
Mobile rendering with capture

- Exploiting mobile device sensors...

Example:
Google Tango
https://www.google.com/atap/projecttango/#project
Mobile rendering with capture

- Exploiting mobile device sensors...

Module 1

see section 5 for more applications of sensor integration
Trends in mobile graphics

• Hardware acceleration for improving frame rates, resolutions and rendering quality
  – Parallel pipelines
  – Real-time ray tracing
  – Multi-rate approaches
**SGRT: Real-time ray tracing**

- Samsung reconfigurable GPU based on Ray Tracing
- **Main key features:**
  - an area-efficient parallel pipelined traversal unit
  - flexible and high-performance kernels for shading and ray generation

Shin et al., *Full-stream architecture for ray tracing with efficient data transmission*, 2014 IEEE ISCAS

Adaptive shading

- Triangles rasterized into coarse fragments that correspond to multiple pixels of coverage
- Coarse fragments are shaded, then partitioned into fine fragments for subsequent per-pixel shading


Mobile rendering with capture

- Exploiting mobile device sensors...

Diagram:
- DATA ACCESS
- Environment
- Scene
- CAPTURE
- RENDERING
- Frame
- DISPLAY
- INTERACTION
- MOBILE DEVICE
Examples: Physical simulations

- Framework for physically and chemically-based simulations of analog alternative photographic processes
- Efficient fluid simulation and manual process running on iPad

Examples: Correcting visual aberrations

- Computational display technology that predistorts the presented content for an observer, so that the target image is perceived without the need for eyewear
- Demonstrated in low-cost prototype mobile devices

Conclusions

• **Heterogeneous applications**
  – driven by bandwidth and processing power

• **Trends**
  – desktop software solutions tend to be ported to the mobile world
    • gaming
    • modelling and 3D animation
    • complex illumination models

• **Sensor integration open new scenarios**
  – examples: live acquisition, mHealth (using sensors and cameras for tracking and processing signals)
Next Session

GRAPHICS DEVELOPMENT FOR MOBILE SYSTEMS
Part 3

Graphics development for mobile systems

Marco Agus, KAUST & CRS4
Mobile Graphics

Heterogeneity

- OS
- Architecture
- Programming languages
- 3D APIs
- IDEs
Mobile Graphics

Heterogeneity

OS
programming languages
IDEs
3D APIs

Mobile Graphics Course – Siggraph Asia 2017
Mobile Graphics

- **OS**
  - Android
  - iOS
  - Windows Phone
  - Firefox OS, Ubuntu Phone, Tizen...

- **Programming Languages**
  - C++
  - Obj-C / Swift
  - Java
  - C# / Silverlight
  - Html5/JS/CSS

- **Architectures**
  - X86 (x86_64): Intel / AMD
  - ARM (32/64bit): ARM + (Qualcomm, Samsung, Apple, NVIDIA,...)
  - MIPS (32/64 bit): Ingenics, Imagination.

- **3D APIs**
  - OpenGL / GL ES
  - D3D / ANGLE
  - Metal / Mantle / Vulkan (GL Next)

- **Cross-development**
  - Qt
  - Marmalade / Xamarin /
  - Muio
  - Monogame / Shiva3D / Unity / UDK4 / Cocos2d-x
Operating Systems
Operating Systems

- Linux based (Qt…)
  - Ubuntu, Tizen, BBOS…

- Web based (Cloud OS)
  - ChromeOS, FirefoxOS, WebOS

- Windows Phone

- iOS (~unix + COCOA)

- Android (JAVA VM)
Development trends

• Hard to follow the trends
  – software does not follow hardware evolution
  – strong market oriented field where finance has strong impact on evolution

• In general, for
  – Mobile phones
    • Market drive towards Android, iOS
  – Tablets
    • Android, iOS, Windows 10
  – Embedded devices
    • Heterogenous (beyond the scopes of this course)

• Here we focus on mobile phones and tablets
Operating Systems

- **Windows 10**
  - Windows development – Visual Studio 2017
    - Good debugging / compiler / integration
  - Great integration and deployment
    - Universal Windows Platform (UWP)
  - API access
    - C#, VB.NET, and C++
  - 3D API
    - D3D
    - OpenGL access through ANGLE
  - Advantages
    - Visual Studio, interoperability with iOS
    - HW is quite selected/homogeneous
  - Disadvantages
    - ~OpenGL wrapper just recently!
Operating Systems

- **iOS**
  - Development under MacOS
    - Xcode – good IDE/debug
    - Clang compiler!
  - API access
    - Objective-C, swift
  - Library programming
    - C++ support
  - Advantages:
    - Homogeneous hardware (biggest issues are resolution related)
    - State-of-the-art CPU/GPU (PowerVR SGX 54X/554, G6400)
    - Good dev tools (Xcode + Clang)
  - Inconvenients:
    - Closed platform
    - Requires iDevice for development/shipment (mostly)
Operating Systems

- Android
  - Development in Eclipse / AndroidStudio
    - Java-based – integrated debugging (non-trivial for NDK)
    - GCC / clang compilers
  - Advantages
    - Wide variety of hardware configurations (CPU/GPU)
    - Java based + C++ as dynamic library (JNI or NDK+NativeActivity)
    - Open source
    - Toolchain provided for Windows/Linux/MacOS (GCC + Clang)
    - Faster access to new hardware / functionality!
  - Inconvenients
    - Heterogeneous device base (hard to target all configurations)
    - Not so integrated IDE -- ~mixed pieces
Operating Systems (comparison)

- App development -- publishing
  - WinPhone & iOS requires less effort for distribution
    - Easy to reach the whole user base
  - Android has a wide variety of configuration that require tuning
    - User base is typically reached in an incremental way (supporting more configs)
    - Many HW configurations (CPU/GPU) give more options to explore 😊
  - Windows has not yet the same market share
    - Variety of configurations
Programming Languages

- **C/C++**
  - Classic, performance, codebase, control
- **Objective C**
  - Bit different style (message based), well-documented API for iOS, mainly COCOA/iOS
- **Java**
  - Android is VM/JIT based, ~portability (API), well-known, extended, codebase
- **C#**
  - VM based, ~Java evolution, (Win, Android, iOS)
- **Swift**
  - Apple new language, simplicity, performance, easy, LLVM-based compilers
- **HTML5/JS**
  - Web technologies, extended, compatibility
- **Perl, Python, Ruby, D, GO (Google), Hack (facebook), …**
  - More options, not so popular?
3D APIs
3D APIs

Cross Platform Challenge

- An explicit API that is also cross-platform needs careful design

One family of GPUs

One OS

One GPU on one OS

Mantle

Direct3D

Metal

OpenGL Next 5.0
3D APIs

- **Direct 3D**
  - 3D API from MS for Win OS (XBOX)
  - ANGLE library provides GL support on top of D3D

- **Mantle**
  - AMD 3D API with Low-level access → **D3D12 | GL_NG**

- **Metal**
  - Apple 3D API with low-level access

- **OpenGL Desktop/ES/WebGL**
  - GL for embedded systems, now in version 3.2
    - GLES3.2 ~ GL4.5

- **GL Next Generation → Vulkan**
  - redesign to unify OpenGL and OpenGL ES into one common API (no backward compatibility)
3D APIs

- **Direct 3D**
  - Games on Windows (mostly) / XBOX
  - Define 3D functionality state-of-the-art
    - OpenGL typically following
    - 3D graphic cards highly collaborative
    - Multithread programming
  - Proprietary – closed source – M$
  - Tested & stable – good support + tools

- **Metal**
  - Apple 3D API with low-level access
  - Much in the way of Mantle?
    - buffer & image, command buffers, sync…
  - Lean & mean → simple + ~flexible
3D APIs

• **Mantle**
  - AMD effort – **low level – direct access** – 3D API
  - Direct control of memory (CPU/GPU) – multithreading done well
    - User-required synchronization
  - **API calls per frame <3k → 100K**
  - Resources: buffer & image 🐨
  - Simplified driver → maintenance (vendors)
    - High level API/Framework/Engines will be developed 🖖
  - Pipeline state
    - shaders + targets (depth/color…) + resources + geometry
  - **Command queues + synchronization**
    - Compute / Draw / DMA(mem. Copy)
  - Bindless – shaders can refer to state resources

  - **OpenGL NEXT seems to move into ‘Mantle direction’**
  - Direct 3D 12 already pursuing low-level access
3D APIs

- **OpenGL (Desktop/ES/WebGL)**
  - Open / research / cross-platform
  - Lagging in front of D3D → Legacy support 😞
    - No more **FIXED PIPELINE (1992)!!** -- scientific visualization…
  - GLSL (2003)...GL 3.1(2009) → deprecation/no fixed pipeline
    - Compatibility profile → legacy again…(till GL 4)
    - Core profile
      - GLSL → shader required
      - VAO
        - group of VBO
        - we need a base VAO for using VBO!
  - Simplifying → VBO + GLSL only!
3D APIs

- OpenGL ES 1.1
  • Fixed pipeline – no glBegin/End – no GL_POLYGON -- VBO
- OpenGL ES 2 (OpenGL 1.5 + GLSL) ~ GL4.1
  • No fixed pipeline (shaders mandatory), ETC1 texture compress..
- OpenGL ES 3 ~ GL4.3
  • Occlusion queries + geometry instancing
  • 32bit integer/float in GLSL
  • Core 3D textures, depth textures, ETC2/EAC, many formats…
  • Uniform Buffer Objects (packed shader parameters)
- OpenGL ES 3.2 ~ GL4.5
  • Compute shaders (atomics, load/store)
  • Separate shader objects (reuse)
  • Indirect draw (shader culling…)
  • NO geometry/tessellation
3D APIs

- **Vulkan**
  - derived from and built upon components of AMD's Mantle API
  - with respect to OpenGL
    - lower level API, more balanced CPU/GPU usage, parallel tasking, work distribution across multiple CPU cores

<table>
<thead>
<tr>
<th>OpenGL</th>
<th>Vulkan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global state machine</td>
<td>No global state</td>
</tr>
<tr>
<td>State tied to context</td>
<td>Common buffer instead of state</td>
</tr>
<tr>
<td>Sequential operations</td>
<td>Multithreaded programming</td>
</tr>
<tr>
<td>Limited control of GPU memory and sync</td>
<td>Explicit control of memory man. and sync</td>
</tr>
<tr>
<td>Extensive error checking</td>
<td>No error checking at runtime</td>
</tr>
</tbody>
</table>
3D APIs

• GPGPU
  – OpenCL
    • On Android it is not much loved
      – Use GPU vendor SDK provided libs
    • On iOS is only accepted for system apps
      – Use old-school GPGPU (fragment shader -> FrameBuffer)

  – Compute shaders
    • GLES 3.2!!! **General solution**!!

  – DirectCompute on D3D
Cross-development

http://www.appian.com/blog/enterprise-mobility-2/are-mobile-platform-choices-limiting-enterprise-process-innovation
Cross platform

- **Unity Mobile (for gaming and VR)**
  - iOS/Android, integration with Tango

- **Unreal Engine 4 (for gaming and VR)**
  - iOS/Android
  - former Unreal Development Kit
  - free usage, payment only for shipping

- **Corona SDK**
  - iOS /Android
  - uses integrated Lua layered on top of C++/OpenGL to build graphic application
  - audio and graphics, cryptography, networking, device information and user input
Cross platform

- **Marmalade**
  - iOS/Android/Windows
  - two main layers
    - low level C API for memory management, file access, timers, networking, input methods (e.g. accelerometer, keyboard, touch screen) and sound and video output.
    - C++ API for higher level functionality for 2D (e.g. bitmap handling, fonts) 3D graphics rendering (e.g. 3D mesh rendering, boned animation), resource management system and HTTP networking.
  - Very successful but dismissing by March 2017

- **EdgeLib**
  - iOS/Android/Windows
  - high performance graphics engine in C++
  - support for 2D graphics, 3D graphics (OpenGL ES), input and sound
Cross platform

• **JMonkey Engine**
  – Android
  – written in Java and using shader technology extensively
  – uses LWJGL as its default renderer (another renderer based on JOGL is available, supporting OpenGL 4)

• **PowerVR**
  – iOS/Android/Windows
  – a cross-platform OS and API abstraction layer, a library of helper tools for maths and resource loading
  – optimized for PowerVR GPUs, with Vulkan support

• **ARM Developer Center**
  – Plenty of tools (computer vision and machine learning, OpenGL ES emulator, texture compression)
Cross-development

- **C++ use case: QtCreator**
  - Qt (supports android, iOS, windows phone, linux, windows, mac)
  - Provides API abstraction for UI, in-app purchases, ~touch input
  - HOWTO (i.e. android):
    - Android SDK
    - Android NDK (native C++ support, toolchain, libraries, GL, CL…)
    - Point environment variables ANDROID_SDK, ANDROID_NDK to folders
    - Create new android project
    - Play!
  - Notes:
    - Go for Qt > 5.4 (touch events were tricky in previous versions)
    - Use QOpenGLWidget instead of QGLWidget
    - Enable touch events on each widget:
      - QWidget::setAttribute(Qt::WA_AcceptTouchEvents);
Mobile Graphics – Development

• Conclusions
  – 1) Native + platform UI ...
    • C++ [any language] → LLVM compiler → target platform
    • Platform Framework front-end → 1 for each platform
    • Performance + flexibility
      • Call native code from platform code (JNI, Object C, ...)
  – 2) Native through framework ...
    • Qt | Marmalade ...
    • C++ code uses framework API
      – Framework API abstracts platform API [N platforms]
      – BUT less flexible integration?
  – 3) Go web → HTML5/JS ...
    • JS code + WebGL
    • ~Free portability (chrome / firefox / IE ... ?)
    • BUT performance is 0.5X at most with asm.js
Next Session

SCALABLE MOBILE VISUALIZATION
Part 4.1

Scalable Mobile Visualization: Introduction

Enrico Gobbetti, CRS4
Scalable mobile visualization

• Goal is high quality interactive rendering of complex scenes…
  – Large data, shading, complex illumination, …
• … on mobile platforms …
  – Mostly smartphones or tablets
  – Similar considerations can apply to other settings (e.g., embedded systems)

• Wide variety of applications
  – Gaming, visualization, cultural heritage…
Mobile platforms scenario

• Typical scalable rendering problem, but with some specific constraints wrt standard (desktop settings)
  ...
• ... screen resolutions are often extremely large (2 – 6 Mpix)
  – Lots of pixels to generate!
• ... mobile 3D graphics hardware is powerful but still constrained
  – Reduced computing powers, memory bandwidths, and amounts of memory wrt desktop graphics systems
  – Limited power supply!
Mobile rendering scenario

• No brute force method applicable
  – Need for “smart methods” to perform interactive rendering
  – Exploit at best reduced rendering power

• Proposed solutions
  – Render only necessary data: adaptive multiresolution
  – Limit required CPU/GPU work: full or partial precomputation
  – Limit data requirements: streaming approaches
  – Exploit at best available bandwidth: data compression
Related Work on mobile visualization

- *(See previous session for details)*
- Remote Rendering
  - ..... 
- Local Rendering
  - Model based
    - Original models
    - Multiresolution models
    - Simplified models
      - Line rendering
      - Point cloud rendering
  - Image based
    - Image impostors
    - Environment maps
    - Depth images
  - Smart shading
  - Volume rendering
Related Work on mobile visualization

• *(See previous session for details)*

• Remote Rendering
  – ..... 

• Local Rendering
  – Model based
    • Original models
    • Multiresolution models
    • Simplified models
      – Line rendering
      – Point cloud rendering
  – Image based
    • Image impostors
    • Environment maps
    • Depth images
  – Smart shading
  – Volume rendering
Scalable Mobile Visualization

• **Big/complex models:**
  – Detailed scenes from modeling, capturing..
    • Output sensitive: adaptive multiresolution
    • Compression / simple decoding

• **Complex rendering**
  – Global illumination
    • Pre-computation
    • Smart shading
  – Volume rendering
    • Compression / simple decoding
Scalable Mobile Visualization. Outline

- Large meshes
- High quality illumination: full precomputation
- High quality illumination: smart computation
- Volume data
Part 4.2

Scalable Mobile Visualization: Large Meshes

Fabio Marton, CRS4
Scalable Mobile Visualization

Extremely Massive 3D Models

1 G Tri
Scalable Mobile Visualization

Itty bitty living space!
A real-time data filtering problem!

• Models of unbounded complexity on limited computers
  – Need for output-sensitive techniques (O(N), not O(K))
  • We assume less data on screen (N) than in model (K →∞)

Storage

View parameters

Limited bandwidth
(network/disk/RAM/CPU/PCIe/GPU/…)

O(K=unbounded) bytes (triangles, points, …)

Projection + Visibility + Shading

Screen

10-100 Hz
O(N=1M-100M) pixels
A real-time data filtering problem!

- Models of unbounded complexity on limited computers
  - Need for output-sensitive techniques (O(N), not O(K))
  - We assume less data on screen (N) than in model (K → ∞)

```
<table>
<thead>
<tr>
<th>Storage</th>
<th>View parameters</th>
<th>Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(K=unbounded) bytes (triangles, points, ...)</td>
<td>Limited bandwidth (network/disk/RAM/CPU/PCIe/GPU/...)</td>
<td>10-100 Hz O(N=1M-100M) pixels</td>
</tr>
</tbody>
</table>
```
Output-sensitive techniques

- At preprocessing time: build MR structure
  - Data prefiltering!
  - Visibility + simplification
  - Compression
- At run-time: selective view-dependent refinement from out-of-core data
  - Must be output sensitive
  - Access to prefiltered data under real-time constraints
  - Visibility + LOD

![Diagram with nodes and edges indicating occlusion and visibility](image-url)
Related work

- **Long history, starting with general solutions**
  - View dependent LOD and progressive streaming [Hoppe 1997]
    - Compute view dependent triangulation each frame -> CPU bound
  - Surface patches [CRS4+ISTI CNR, SIGGRAPH’04]
    - Effective in terms of speed
    - Require non-trivial data structures and techniques for decompression
  - General solutions available for Desktop environments [Cignoni et al, 2005, Yoon et al. 2008]

- **Mesh compression – MPEG-4 [Jovanova et al. 2008]**

- **Light 3D model rendering [MeshPad, PCL]**

- **Gigantic point clouds on mobile devices [Balsa et al. 2012]**

- … and much more
Our Contributions: chunked multiresolution structures

• Efficient view-dependent meshes
  – Approximate original surface
  – Seamless

• Mix and match chunks
  – Amortize CPU work!

• Two approaches
  – Fixed coarse subdivision
    • Adaptive QuadPatches
  – Adaptive coarse subdivision
    • Compact Adaptive TetraPuzzles
Adaptive Quad Patches
Simplified Streaming and Rendering for Mobile & Web

• Represent models as fixed number of multiresolution quad patches
  – Image representation allows component reuse!
  – Natural multiresolution model inside each patch
  – Adaptive rendering handled totally within shaders!

• Works with topologically simple models
Related work Adaptive Quad Patches

- **Geometry images** [Gu et al. 2002]
  - Exploit current GPU capabilities / optimized libraries for compression and streaming of images

- **Quad remeshing**
  - Single-disk parametrization [Floater and Hormann 2005]
  - Base mesh to parametrize the model [Petroni et al. 2010]

- **Detail rendering**
  - GPU raycasting [Oliveira et al. 2000]
  - Displacement mapping in GPU [Shiue et al. 2005]
AQP Approach

• Models partitioned into fixed number of quad patches
  – Geometry encoded as detail with respect to the 4 corners interpolation

• For each quad: 3 multiresolution pyramids
  – Detail geometry
  – Normals
  – Colors

• Data encoded as images
  – Exploit .png (lossless compression)

• Ensure connectivity
  – Duplicated boundary information
Pre-processing (Reparameterization)

- **Generate clean manifold triangle mesh**
  - Poisson reconstruction [Kazhdan et al. 2006]
  - Remove topological noise
    - Discard connected components with too few triangles

- **Parameterize the mesh on a quad-based domain**
  - Isometric triangle mesh parameterization
    - Abstract domains [Pietroni et al. 2010]
    - Remap into a collection of 2D square regions

- **Resample each quad from original geometry**
  - Associates to each quad a regular grid of samples (position, color and normal)
Pre-processing (Multiresolution)

• Collection of variable resolution quad patches
  – Coarse representation of the original model

• Multiresolution pyramids
  – Detail geometry
  – Color
  – Normals

• Shared border information
  – Ensure connectivity
Adaptive rendering

• **CPU Lod Selection**
  - Different quad LOD, but must agree on edges
    • Quad LOD = max edge LOD (available)
  - If LOD not available post asynchronous request, use finest

• **GPU drawing with Vertex Shader**
  - Quad corners
  - 1 VBO per resolution level reused (u,v)
  - texture mipmaps of
    • Displacements, Normals, Colors
  - Texture with edge LODs (snap)
Rendering example

Patches

Levels

Shading
Adaptive rendering

1. CPU LOD Selection
   - Find edge LODs
   - Quad LOD = max edge LODs
   - If data available use it, otherwise
     - Query data for next frames
     - Use best available representation
   - Send VBO with regular grid (1 for each LOD)

2. GPU: Vertex Shader
   - Snap vertices on edges (match neighbors)
   - Base position = corner interpolation \((u,v)\)
   - Displace VBO vertices
     - normal + displacement (dequantized)

3. GPU: Fragment Shader
   - Texturing & Shading
## Results

<table>
<thead>
<tr>
<th>St. Matthew</th>
<th>374 M Tri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg bps</td>
<td>24.3 (6.3 + 9.5 + 8.5) (pos + color + normal)</td>
</tr>
<tr>
<td>Pixel Accuracy</td>
<td>1</td>
</tr>
<tr>
<td>FPS avg</td>
<td>37</td>
</tr>
<tr>
<td>FPS min</td>
<td>13</td>
</tr>
<tr>
<td>ADSL 8Mbps refine time</td>
<td>2s for model from scratch</td>
</tr>
</tbody>
</table>
Adaptive Quad Patches Conclusions

• Effective creation and distribution system
  – Fully automatic
  – Compact, streamable and renderable 3D model representations
  – Low CPU overhead
  – WebGL
    • Desktop
    • Mobile

• Limitations
  – Closed objects with large components
  – Visual approximation (lossy)
  – Explore more aggressive compression techniques
  – Occlusion culling
  – More sophisticated shading/shadowing techniques

• Next: More general solution based on full multiresolution structure
Compact Adaptive TetraPuzzles
Adaptive multiresolution solution with compression-domain rendering

- Represent models as variable number of multiresolution surface patches embedded in a hierarchy of tetrahedra
  - Regular conformal hierarchy of tetrahedra spatially partitions input mesh
    * Mesh fragments at different resolutions associated to implicit diamonds
  - Fully adaptive and seamless 3D mesh structure with local quantization
    * Geometry clipped against containing tetrahedra
    * Barycentric coordinates used for local tetrahedra geometry reparameterization
  - GPU friendly compact data representation
- Works with general surface models
Compact Adaptive Tetra Puzzles

Triangle soup

Partitioned input model

Tetrahedra hierarchy

Database

Simplified representations

Tetrahedra hierarchy

Partitioning

Merging & Simplification

Encoding & Compression

Simplified representations

Tetrahedra hierarchy

Database

Partitioned input model

Triangle soup
Related work (Compression)

**Topology coding**
- Theoretical minimum [Rossignac 2001]
  - 1.62 bits/triangle, 3.24 bits/vertex
- 8 bpt/16 bpv [Chhugani et al. 2007]
  - HW-implementation
- 5 bpt/10 bpv [Meyer et al. 2012]
  - CUDA implementation

**Attribute quantization**
- Global position quantization [Lee et al. 2009]
- Local quantization techniques [Lee et al. 2010]
- Normal compression using octahedral parametrization [Meyer et al. 2010]

**Our goal is to balance compression rate and decoding+rendering performance by using a GPU-friendly compact representation**
Overview

• Construction
  – Start with hires triangle soup
  – Partition model
  – Construct non-leaf cells by bottom-up recombination and simplification of lower level cells
  – Assign model space errors to cells

• Rendering
  – Refine graph
  – Render selected precomputed cells

Ensure continuity → Shared information on borders
Preprocessing

- **Geometry clipped against containing tetrahedra**
- **Vertices: tetrahedra barycentric coordinates**
  - \[ P_{\text{barycentric}} = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3 + \lambda_4 P_4 \]
- **Seamless local quantization**
  - Inner vertices (I): 4 corners
  - Face vertices (F): 3 corners
  - Edge vertices (E): 2 corners
- **GPU friendly compact data representation**
  - 8 bytes = position (3 bytes) + color (3 bytes) + normal (2 bytes)
  - Normals encoded with the octahedron approach [Meyer et al. 2012]
- **Further compression with entropy coding**
  - Exploiting local data coherence
Rendering process

- Extract view dependent diamond cut (CPU)
- Request required patches to server
  - Asynchronous multithread client
  - Apache 2 based server (data repository, no processing)
- CPU entropy decoding of each patch
- For each node (GPU Vertex Shader):
  - VBO with barycentric coordinates, normals and colors (64 bpv)
  - Decode position: \( P = MV \times [C0 \ C1 \ C2 \ C3] \times [Vb] \)
    - \( Vb \) is the vector with the 4 barycentric coords
    - \( C0..C3 \) are tetrahedra corners
  - Decode normal from 2 bytes encoding [Meyers et al. 2012]
  - Use color coded in RGB24
Results

- **Input Models**
  - St. Matthew 374 MTri
  - David 1GTri

- **Compression**
  - 40 to 50 bits/vertex

- **Streaming full screen view**
  - 30s on wireless,
  - 45s on 3G
  - David 14.5MB (1.1 Mtri)
  - St. Matthew 19.9MB (1.8 Mtri)

---

<table>
<thead>
<tr>
<th>Rendering</th>
<th>iPad gen3</th>
<th>iPhone 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel tolerance</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Triangle throughput</td>
<td>30 Mtri/s</td>
<td>2.8 Mtri/s</td>
</tr>
<tr>
<td>FPS avg</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>FPS refined views</td>
<td>15</td>
<td>2.8</td>
</tr>
<tr>
<td>Triangle Budget</td>
<td>2 M</td>
<td>1 M</td>
</tr>
</tbody>
</table>
Conclusions: Compact ATP

• Generic gigantic 3D triangle meshes on common handheld devices
  – Compact, GPU friendly, adaptive data structure
    • Exploiting the properties of conformal hierarchies of tetrahedra
    • Seamless local quantization using barycentric coordinates
  – Two-stage CPU and GPU compression
    • Integrated into a multiresolution data representation

• Limitations
  – Requires coding non-trivial data structures
  – Hard to implement on scripting environments
Conclusions: large meshes

• Various solutions for large meshes

• **Constrained solution: Adaptive Quad Patches**
  – Simple and fast
  – Good compression
  – Works on topologically simple models

• **General solution: Compact Adaptive Tetra Puzzles**
  – Compact data representation
  – More complex code
Part 4.3

Scalable Mobile Visualization: Introduction to complex lighting

Enrico Gobbetti, CRS4
Complex scenes

• We have seen how to deal with complex meshes $O(G_{tri})$
  – Similar solutions for point clouds…

• Problem tackled was size
  – Solution proposed: adaptive multiresolution chunk-based approaches
  – Various optimized solutions to select chunks, compose them, …

• Rendering was simple, though
  – One pass streaming, direct illumination

• How to deal with more complex illumination and shading?
Complex scenes

• Complex illumination/shading introduce data and computation problems
  – Non-local effects (global illumination, shadows, …) require scattered information
  – Illumination/shading is costly (CPU/GPU time) and requires data-intensive algorithms

• Proposed solutions in the mobile world
  – Full precomputation
    • Images computed off-line
    • Removes real-time timing constraints, but introduces other problems (which images to compute? How to navigate in an image-based scene?)
  – Smart computation
    • Partial precomputation of some intermediate results, approximation tricks
    • Not general solution but improves quality!

• Next session illustrates examples of full/smart computation in mobile graphics
Part 4.4

Scalable Mobile Visualization:
Full precomputation of complex lighting

Fabio Marton, CRS4
Ubiquitous exploration of scenes with complex illumination

- **Real-time requirement: ~30Hz**
  - Difficulties handling complex illumination on mobile/web platforms with current methods

- **Image-based techniques**
  - Constraining camera movement to a set of fixed camera positions
  - Enable pre-computed photorealistic visualization

- **Explore-Maps: technique for**
  - Scene representation as set of probes and arcs
  - Precomputed rendering for probes and transitions
Scene Discovery

- **ExploreMaps**: Automatic best view/best path methods for generating
  - Set of probes providing full model coverage
    - Probe = 360° panoramic point of view
  - Set of arcs connecting probes
    - Enable full scene navigation

Di Bendeetto et al. Eurographics 2014

**ExploreMaps**: Efficient Construction and Ubiquitous Exploration of Panoramic View Graphs of Complex 3D Environments.
Best viewpoints computation

- **Position set of probes inside the scene**
  - Probes provide a 360 degree view
  - Greedy algorithm that places probes at the barycenter of newly seen geometry until all the scene is visible
  - Final clustering pass reduces number of probes

Coverage optimization, by moving to the barycenter of seen geometry
Best path computation

• Connect probes which have a common visible region
  – Creates a graph of probes

• For each pair of mutually visible probe
  – Create first path going through the closest point in the mutually visible region
  – Optimize and smooth the path using a mass-spring system
Precomputation of probe images

- Compute panoramic views for probes and frames of transition arcs
  - Photorealistic rendering (using Blender 2.68a)
    - panoramic views both for probes and transition arcs
  - $1024^2$ probe panoramas
  - $256^2$ transition video panoramas
  - 32 8-core PCs,
  - Rendering times ranging from 40 minutes to 7 hours/model
## Explore Maps – Processing Results

<table>
<thead>
<tr>
<th>Input #</th>
<th>Museum</th>
<th>Sponza</th>
<th>Sibenik</th>
<th>Lighthouse</th>
<th>Lost Empire</th>
<th>Medieval Town</th>
<th>German Cottage</th>
<th>Neptune</th>
</tr>
</thead>
<tbody>
<tr>
<td>#verts</td>
<td>1,468,140</td>
<td>262,267</td>
<td>69,853</td>
<td>48,940</td>
<td>157,136</td>
<td>14,865</td>
<td>79,400</td>
<td>2,227,359</td>
</tr>
<tr>
<td>#edges</td>
<td>70</td>
<td>36</td>
<td>92</td>
<td>57</td>
<td>74</td>
<td>78</td>
<td>140</td>
<td>79</td>
</tr>
<tr>
<td>#clusters</td>
<td>17</td>
<td>10</td>
<td>21</td>
<td>17</td>
<td>25</td>
<td>30</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>#paths</td>
<td>127</td>
<td>29</td>
<td>58</td>
<td>81</td>
<td>206</td>
<td>222</td>
<td>102</td>
<td>93</td>
</tr>
<tr>
<td>Time (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exploration</td>
<td>154</td>
<td>23</td>
<td>63</td>
<td>15</td>
<td>41</td>
<td>34</td>
<td>163</td>
<td>38</td>
</tr>
<tr>
<td>Clustering</td>
<td>17</td>
<td>3</td>
<td>27</td>
<td>8</td>
<td>13</td>
<td>14</td>
<td>118</td>
<td>14</td>
</tr>
<tr>
<td>Synthesis</td>
<td>144</td>
<td>35</td>
<td>449</td>
<td>453</td>
<td>284</td>
<td>395</td>
<td>427</td>
<td>279</td>
</tr>
<tr>
<td>Path</td>
<td>7</td>
<td>1</td>
<td>31</td>
<td>12</td>
<td>22</td>
<td>80</td>
<td>23</td>
<td>13</td>
</tr>
<tr>
<td>Path smoothing</td>
<td>3.012</td>
<td>122</td>
<td>81</td>
<td>89</td>
<td>482</td>
<td>199</td>
<td>185</td>
<td>150</td>
</tr>
<tr>
<td>Thumbn.</td>
<td>11</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Thumbn. pos</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>3,347</td>
<td>189</td>
<td>659</td>
<td>583</td>
<td>854</td>
<td>736</td>
<td>925</td>
<td>501</td>
</tr>
<tr>
<td>Storage (MB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probes</td>
<td>59</td>
<td>28</td>
<td>72</td>
<td>59</td>
<td>86</td>
<td>103</td>
<td>79</td>
<td>43</td>
</tr>
<tr>
<td>Paths</td>
<td>248</td>
<td>146</td>
<td>113</td>
<td>159</td>
<td>371</td>
<td>376</td>
<td>390</td>
<td>120</td>
</tr>
</tbody>
</table>
Interactive Exploration

• UI for Explore Maps
  – WebGL implementation + JPEG + MP4
  – Panoramic images: probes + transition path

• Closest probe selection
  • Path alignment with current view

• Thumbnail goto
  – Non-fixed orientation
Conclusion: Interactive Exploration

- **Interactive exploration of complex scenes**
  - Web/mobile enabled
  - Precomputed rendering
    - State-of-the-art Global Illumination
  - Graph-based navigation → guided exploration

- **Limitations**
  - Constrained navigation
    - Fixed set of camera positions
  - Limited interaction
    - Exploit panoramic views on paths → less constrained navigation

- **Next part of the talk:**
  - A dynamic solution for complex illumination with smart computation
Part 4.5

Scalable Mobile Visualization:
Smart precomputation for complex lighting

Pere-Pau Vázquez, UPC
High quality illumination

• Consistent illumination for AR
• Soft shadows
• Deferred shading
• Ambient Occlusion
Consistent illumination for AR

- High-Quality Consistent Illumination in Mobile Augmented Reality by Radiance Convolution on the GPU [Kán, Unterguggenberger & Kaufmann, 2015]

- Goal
  - Achieve realistic (and consistent) illumination for synthetic objects in Augmented Reality environments
Consistent illumination for AR

• Overview
  – Capture the environment with the mobile
  – Create an HDR environment map
  – Convolve the HDR with the BRDF’s of the materials
  – Calculate radiance in realtime
  – Add AO from an offline rendering as lightmaps
  – Multiply with the AO from the synthetic object
Consistent illumination for AR

• Capture the environment with the mobile
  – Rotational motion of the mobile
    • In yaw and pitch angles to cover all sphere directions
  – Images accumulated to a spherical environment map

• HDR environment map constructed while scanning
  – Projecting each camera image
    • According to the orientation and inertial measurement of the mobile
  – Low dynamic range imaging is transformed to HDR
    • Camera uses auto-exposure
      – Two overlapping images will have slightly different exposure
  – Alignment correction based on feature matching
  – All in the device
Consistent illumination for AR

• Convolve the HDR with the BRDF’s of the materials
  – Use MRT to support several convolutions at once
  – Assume distant light
  – One single light reflection on the surface
  – Scene materials assumed non-emissive
  – Use a simplified rendering equation

• Weight with AO (obtained offline)
  – Built for real and synthetic objects
  – Need the geometry of the scene
    • Use a proxy geometry for the objects of the real world
    • Cannot be simply done on the fly
Consistent illumination for AR

- Results

Without AO

With AO

Images courtesy of Peter Kán
Consistent illumination for AR

• Performance

<table>
<thead>
<tr>
<th>3D model</th>
<th># triangles</th>
<th>Framerate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflective cup</td>
<td>25.6K</td>
<td>29 fps</td>
</tr>
<tr>
<td>Teapot</td>
<td>15.7K</td>
<td>30 fps</td>
</tr>
<tr>
<td>Dragon</td>
<td>229K</td>
<td>13 fps</td>
</tr>
</tbody>
</table>

• Limitations
  – Materials represented by Phong BRDF
  – AO and most shading (e.g. reflection maps) is baked
Soft shadows using cubemaps

• Efficient Soft Shadows Based on Static Local Cubemap [Bala & Lopez Mendez, 2016]

• Goal
  – Soft shadows in realtime

Taken from https://community.arm.com/graphics/b/blog/posts/dynamic-soft-shadows-based-on-local-cubemap
Soft shadows using cubemaps

- **Overview**
  - Create a local cube map
    - Offline recommended
    - Stores color and transparency of the environment
    - Position and bounding box
      - *Approximates the geometry*
    - Local correction
      - Using proxy geometry
  - Apply shadows in the fragment shader
Soft shadows using cubemaps

• Generating shadows
  – Fetch texel from cubemap
    • Using the fragment-to-light vector
    • Correct the vector before fetching
      – Using the scene geometry (bbox) and cubemap creation position
        » To provide the equivalent shadow rays
  – Apply shadow based on the alpha value
  – Soften shadow
    • Using mipmapping and addressing according to the distance
Soft shadows using cubemaps

• Conclusions
  – Does not need to render to texture
    • Cubemaps must be pre-calculated
  – Requires reading multiple times from textures
  – Stable
    • Because cubemap does not change

• Limitations
  – Static, since info is precomputed
Physically-based Deferred Rendering

• Physically Based Deferred Shading on Mobile [Vaughan Smith & Einig, 2016]

• Goal:
  – Adapt deferred shading pipeline to mobile
  – Bandwidth friendly
  – Using Framebuffer Fetch extension
    • Avoids copying to main memory in OpenGL ES
Physically-based Deferred Rendering

- **Overview**
  - Typical deferred shading pipeline
Physically-based Deferred Rendering

- Main idea: group G-buffer, lighting & tone mapping into one step
  - Further improve by using Pixel Local Storage extension
    - G-buffer data is not written to main memory
    - Usable when multiple shader invocations cover the same pixel
  - Resulting pipeline reduces bandwidth
Physically-based Deferred Rendering

• Two G-buffer layouts proposed
  – Specular G-buffer setup (160 bits)
    • Rgb10a2 highp vec4 light accumulation
    • R32f highp float depth
    • 3 x rgba8 highp vec4: normal, base color & specular color
  – Metallicness G-buffer setup (128 bits, more bandwidth efficient)
    • Rgb10a2 highp vec4 light accumulation
    • R32f highp float depth
    • 2 x rgba8 highp vec4: normal & roughness, albedo or reflectance metallicness
Physically-based Deferred Rendering

• Lighting
  – Use precomputed HDR lightmaps to represent static diffuse lighting
    • Shadows & radiosity
  – Can be compressed with ASTC (supports HDR data)
    • PVRTC, RGBM can also be used for non HDR formats
  – Geometry pass calculates diffuse lighting
  – Specular is calculated using Schlick’s approximation of Fresnel factor
Physically-based Deferred Rendering

• **Results (PowerVR SDK)**
  – Fewer rendering tasks
    • meaning that the G-buffer generation, lighting, and tonemapping stages are properly merged into one task.
    • reduction in memory bandwidth
      – 53% decrease in reads and a 54% decrease in writes

• **Limitations**
  – Still not big frame rates
Ambient Occlusion in mobile

• Optimized Screen-Space Ambient Occlusion in Mobile Devices [Sunet & Vázquez, Web3D 2016]

• Goal: Study feasibility of real time AO in mobile
  – Analyze most popular AO algorithms: Crytek’s, Alchemy’s, Nvidia’s Horizon-Based AO (HBAO), and Starcraft II (SC2)
  – Evaluate their AO pipelines step by step
  – Design architectural improvements
  – Implement and compare
Ambient Occlusion in mobile

- **Ambient Occlusion. Simplification of rendering equation**
  - The surface is a perfect diffuse surface (BRDF constant)
  - Light potentially reaches a point $p$ equally in all directions
    - But takes into account point’s visibility

\[ L_o(p, \omega_o) = \frac{1}{\pi} \int_{\Omega} \rho(d(p, \omega_i)) \cos \theta_i \ d\omega_i \]

\[ \rho(d) = \begin{cases} 
  f(d) \in [0, 1] & d < \text{threshold} \\
  0 & \text{otherwise} 
\end{cases} \]
Ambient Occlusion in mobile

• AO typical implementations
  – Precomputed AO: Fast & high quality, but static, memory hungry
  – Ray-based: High quality, but costly, visible patterns…
  – Geometry-based: Fast w/ proxy structures, but lower quality, artifacts/noise…
  – Volume-based: High quality, view independent, but costly

  – Screen-space:
    • Extremely fast
    • View-dependent
    • [mostly] requires blurring for noise reduction
    • Very popular in video games (e.g. Crysis, Starcraft 2, Battlefield 3…)
Ambient Occlusion in mobile

• Screen-space AO:
  – Approximation to AO implemented as a screen-space post-processing
    • ND-buffer provides coarse approximation of scene's geometry
    • Sample ND-buffer to approximate (estimate) ambient occlusion instead of shooting rays
Ambient Occlusion in mobile

• SSAO pipeline
  1. Generate ND (normal + depth, OpenGL ES 2) or G-Buffer (ND + RGB…, OpenGL ES 3.+)  
  2. Calculate AO factor for visible pixels  
     a. Generate a set of samples of positions/vectors around the pixel to shade.  
     b. Get the geometry shape (position/normal…)  
     c. Calculate AO factor by analyzing shape…  
  3. Blur the AO texture to remove noise artifacts  
  4. Final compositing
Ambient Occlusion in mobile

- **Optimizations. G-Buffer storage**
  - G-Buffer with less precision (32, 16, 8)
    - 8 not enough
    - 16 and 32 similar quality
  - Normal storage (RGB vs RG)
    - RGB normals are faster
Ambient Occlusion in mobile

- **Optimizations. Sampling**
  - AO samples generation (disc and hemisphere)
    - Desktops use up to 32
    - With mobile, 8 is the affordable amount
      - Pseudo-random samples produces noticeable patterns
  - Our proposed solution
    - Compute sampling patterns offline
      - 2D: 8-point Poisson disc
      - 3D: 8-point cosine-weighted hemisphere (Malley’s approach, as in [Pharr and Humphreys, 2010])
    - Scaling and rotating the resulting pattern ([Chapman, 2011])
    - Predictable, reproducible, robust
Ambient Occlusion in mobile

- **Optimizations. Getting geometry positions**
  - Transform samples to 3D
    - Inverse transform vs similar triangles
      - Precision for speed
    - Similar triangles are faster
  - Storing depth vs storing 3D positions in G-Buffer
    - Trades bandwidth for memory
    - Depth slightly better
    - Better profile for the application
Ambient Occlusion in mobile

- **Optimizations. Banding & Noise**
  - Fixed sampling pattern produces banding (left)
  - Random sampling reduces banding but adds noise (middle)
  - SSAO output is typically blurred to remove noise (right)
    - But blurs edges
Ambient Occlusion in mobile

- **Optimizations. Banding & Noise**
  - User bilateral filter instead
    - Works better
    - Improve timings with separable filter

\[
BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p - q||) G_{\sigma_r}(||I_q - I_p||) I_q
\]

\[
G_{\sigma}(x) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)
\]
Ambient Occlusion in mobile

- Optimizations. Progressive AO
  - Amortize AO throughout many frames
Ambient Occlusion in mobile

• Optimizations
  – Naïve improvement: Reduce the calculation to a portion of the screen
    • Mobile devices have a high PPI resolution
    • Reduction improves timings dramatically while keeping high quality
  – Typical reduction:
    • Offscreen render to 1/4th of the screen
    • Scale-up to fill the screen
Ambient Occlusion in mobile

• Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Optimized (not progressive)</th>
<th>Optimized + progressive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starcraft 2</td>
<td>17.8%</td>
<td>38.5%</td>
</tr>
<tr>
<td>HBAO</td>
<td>25.6%</td>
<td>39.2%</td>
</tr>
<tr>
<td>Crytek</td>
<td>23.4%</td>
<td>35.0%</td>
</tr>
<tr>
<td>Alchemy</td>
<td>24.8%</td>
<td>38.2%</td>
</tr>
</tbody>
</table>
Ambient Occlusion in mobile

• Conclusions
  – Developed an optimized pipeline for mobile AO
    • Analyzed the most popular AO techniques
      – Improved several important steps of the pipeline
      – Proposed some extra contributions (e.g. progressive AO)
    • Achieved realtime framerates with high quality
    • Developed techniques can be used in WebGL
  – Future Work
    • Further improvement of the pipeline
    • Developing “Homebrew” method
      – With all known improvements
      – Some extra tricks
      – Not ready for prime time yet
Part 4.5

**Scalable Mobile Visualization:**

**Volumetric Data**

Pere-Pau Vázquez, UPC
Rendering Volumetric Datasets

- Introduction
- Challenges
- Architectures
- GPU-based ray casting on mobile
- Conclusions
Rendering Volumetric Datasets

Capturing

Rendering

3D texture

GPU-based ray casting

Output
Rendering Volumetric Datasets

• **Introduction**
  – Volume datasets
    • Sizes continuously growing (e.g. $>1024^3$)
    – Complex data (e.g. 4D)
  – Rendering algorithms
    • GPU intensive
    • State-of-the-art is ray casting on the fragment shader
  – Interaction
    • Edition, inspection, analysis, require a set of complex manipulation techniques
Rendering Volumetric Datasets

• Desktop vs mobile
  – Desktop rendering
    • Large models on the fly
    • Huge models with the aid of compression/multiresolution schemes
  – Mobile rendering
    • Standard sizes (e.g. $512^3$) still too much for the mobile GPUs
    • Rendering algorithms GPU intensive
      – State-of-the-art is GPU-based ray casting
    • Interaction is difficult on a small screen
      – Changing TF, inspecting the model…
Rendering Volumetric Datasets

• **Challenges on mobile:**
  – Memory:
    • Model does not fit into memory
      – Use client server approach / compress data
  – GPU capabilities:
    • Cannot use state of the art algorithm (e.g. no 3D textures)
      – Texture arrays
  – GPU horsepower:
    • GPU unable to perform interactively
      – Progressive rendering methods
  – Small screen
    • Not enough details, difficult interaction
Rendering Volumetric Datasets

- **Mobile architectures**
  - Server-based rendering
  - Hybrid approaches
  - Pure mobile rendering
  - Server-based and hybrid rely on high bandwidth communication
Rendering Volumetric Datasets

• **Pure mobile rendering**
  – Move all the work to the mobile
  – Nowadays feasible

• **Direct Volume Rendering on mobile. Algorithms**
  – Slices
  – 2D texture arrays
  – 3D textures
Rendering Volumetric Datasets

• Slices
  – Typical old days volume rendering
    • Several quality limitations
    • Subsampling & view change
  – Improvement: Oblique slices [Kruger 2010]
Rendering Volumetric Datasets

• **2D texture arrays + texture atlas [Noguera et al. 2012]**
  - Simulate a 3D texture using an array of 2D textures
  - Implement GPU-based ray casting
    • High quality
    • Relatively large models
    • Costly
    • Cannot use hardware trilinear interpolation
Rendering Volumetric Datasets

- 2D texture arrays + texture atlas
Rendering Volumetric Datasets

• 2D texture arrays + compression [Valencia & Vázquez, 2013]
  – Increase the supported sizes
  – Increase framerates

<table>
<thead>
<tr>
<th>Compression format</th>
<th>Compression ratio</th>
<th>RBA format</th>
<th>RGBA format</th>
<th>GPU support</th>
<th>Overall performance</th>
<th>Overall quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETC1</td>
<td>4:1</td>
<td>Yes</td>
<td>No</td>
<td>All GPUs</td>
<td>Good (RC)</td>
<td>Good</td>
</tr>
<tr>
<td>PVRTC</td>
<td>8:1 and 16:1</td>
<td>Yes</td>
<td>Yes</td>
<td>PowerVR</td>
<td>Not so good</td>
<td>Bad</td>
</tr>
<tr>
<td>ATITC</td>
<td>4:1</td>
<td>Yes</td>
<td>Yes</td>
<td>Adreno</td>
<td>Good (RC)</td>
<td>Good</td>
</tr>
</tbody>
</table>
Rendering Volumetric Datasets

- 2D texture arrays + compression
  - ATITC: improves performance from 6% to 19%. With an average of 13.1% and a low variance of performance.
  - ETC1(-P): improves performance from 6.3% to 69.5%. With an average of 32.6% and the highest variance of performance.
  - PVRTC-4BPP: improves performance from 4.7% and 36.% and PVRTC-2BPP: from 9,5% to 36,5%. The average performance of both methods is ~15% with high variance.
Rendering Volumetric Datasets

• 2D texture arrays + compression
  – Ray-casting: gain performance in average of 33%.
  – Slice-based: gain performance in average of 8%.
  – Ray-casting frame rates are better in all cases compared to slice-based.
Rendering Volumetric Datasets

- 2D texture arrays + compression
Rendering Volumetric Datasets

- 2D texture arrays + compression

Uncompressed  Compressed with PVRTC-4BPP  Compressed with PVRTC-2BPP
Rendering Volumetric Datasets

• **3D textures [Balsa & Vázquez, 2012]**
  – Allow either 3D slices or GPU-based ray casting
  – Initially, only a bunch of GPUs sporting 3D textures (Qualcomm’s Adreno series >= 200)
  – Performance limitations (data: $256^3$ – screen resol. 480x800)
    • 1.63 for 3D slices
    • 0.77 fps for ray casting
Rendering Volumetric Datasets

2D slices comparison

- Nexus ONE
- HTC Desire
- HTC Desire HD
- HTC Desire Z
- Samsung Galaxy S2
- Advent Vega
- LG Optimus 2X
- Samsung Galaxy S
Rendering Volumetric Datasets

• 2D slices
Rendering Volumetric Datasets

- 2D slices vs 3D slices vs raycasting
Rendering Volumetric Datasets

- Using Metal on an iOS device [Schiewe et al., 2015]

Taken from [Schiewe et al., 2015]
Volume data. GPU ray casting on mobile

- **Using Metal on an iOS device [Schiewe et al., 2015]**
  - Standard GPU-based ray casting
  - Provides low level control
  - Improved framerate (2x, to a maximum of 5-7 fps) over slice-based rendering
  - Models noticeably smaller than available memory (max. size was $256^2 \times 942$)
Rendering Volumetric Datasets

- Challenges: Transfer Function edition
Rendering Volumetric Datasets

- Challenges: Transfer Function edition
Rendering Volumetric Datasets

• Conclusion
  – Volume rendering on mobile devices possible but limited
    • Can use adaptive rendering (half resolution when interacting)
  – 3D textures in core GLES 3.0
    • Still limited performance (~7fps…)
  – Interaction still difficult
  – Client-server architecture still alive
    • Can overcome data privacy/safety & storage issues
    • Better 4G-5G connections
    • …
Next Session

MOBILE METRIC CAPTURE AND RECONSTRUCTION
Part 5.1

Mobile Metric Capture & Reconstruction: Introduction

Enrico Gobbetti, CRS4
Computer vision and mobile applications

- **Digital photos** (auto enhance)
- **Face detection**
- **Biometrics** (fingerprints)
- **HDR**
- **Panoramic photos** (autostitch)
- **Image Search**
- **Visual Search**
  - Landmark recognition
- **Augmented Reality**
- **Material Capture**
- **3D capture**
- **VSLAM**

Timeline:
- **1990**
- **2000**
- **2010**
Computer vision and mobile applications

- Mostly 2D
  - Image enhancement
  - Image stitching
  - Image matching
  - Object detection
  - Texture classification
  - Activity recognition
  - ...

- Mostly 3D
  - Camera localization
  - Pose estimation
  - 3D shape recovery
  - 3D scene reconstruction
  - Material/appearance recovery
  - Augmented reality
  - ...

Applications made possible by specific features of mobile devices!

- Features
  1. Mobility
  2. Camera
  3. Active light
  4. Non-visual sensors
  5. Processing power
  6. Connectivity
  7. Display
Features (1/7): Mobility

- **Consumer**
  - Smartphones
  - Tablets

- **Embedded**
  - Autonomous driving
  - Assistive technologies

- **Specific**
  - Drones
  - Robots
Features (1/7): Mobility

• **Consumer**
  – Smartphones
  – Tablets

• **Embedded**
  – Autonomous driving
  – Assistive technologies

• **Specific**
  – Drones
  – Robots

On-site applications / Personal applications / Motion and/or location taken into account / Embedded solutions
Features (2/7): High-res/flexible camera

• Common features
  – High resolution and good color range (>12 MP, HDR)
  – Small sensors (similar to point and shoot cameras – approx. 1/3”)
  – High video resolution and frame rate (4K at 30fps)

• Wide variety of field of views
  – standard, fisheye, spherical

• Specialized embedded cameras…
  – Better lenses and sensors…
Features (2/7): High-res/flexible camera

• **Common features**
  - High resolution and good color range (>12 MP, HDR)
  - Small sensors (similar to point and shoot cameras – approx. 1/3”)
  - High video resolution and frame rate (4K at 30fps)

• **Wide variety of fields of view**
  - standard, fisheye

• **Specialized embedded cameras**
  - Better lenses and sensors…

Visual channel is the primary one
Computational photography
Apps analyze/use snapshots or videos
Features (3/7): Active lighting

• All smartphones have a flashlight
  – LED source at fixed distance from camera

• Custom devices have integrated emitters
  – Google TANGO / Microsoft Kinect
    • Integrated depth sensor

• Leads to specialized capture procedures
Features (3/7): Active lighting

- All smartphones have a flashlight
  - LED source at fixed distance from camera
- Custom devices have integrated emitters
  - Google TANGO / Microsoft Kinect
    - Integrated depth sensor
- Leads to specialized capture procedures

Specialized capture procedures exploiting synchronization of illumination and visual sensing

Features (4/7): Non-visual sensors

- **Absolute reference**
  - GPS / A-GPS
    - Mainly for outdoor applications
  - Magnetometer
    - Enable compass implementation
    - Often inaccurate for indoor applications

- **Relative reference**
  - Accelerometer
    - Variable accuracy (sensitive to temperature)
    - Good metric information for small scale scene
  - Gyroscope
    - Very good accuracy for device relative orientation

- **Synced with camera!**
Features (4/7): Non-visual sensors

- **Absolute reference**
  - GPS / A-GPS
    - Mainly for outdoor applications
  - Magnetometer
    - Enable compass implementation
    - Often inaccurate indoors
- **Relative reference**
  - Accelerometer
    - Variable accuracy (sensitive to temperature)
    - Good metric information for small scale scenes
  - Gyroscope
    - Very good accuracy for device relative orientation
    - Synced with camera!

Data fusion!

Features (5/7): Processing power

- Growing performance of mobile CPU+GPU  
  - (see previous sections)
- Capable to execute computer vision pipeline on mobile device  
  - i.e. OpenCV for Android
- Some limitations due to power consumption
Features (5/7): Processing power

- Growing performance of mobile CPU+GPU
  
  - (see previous section)

- Capable to execute computer vision pipeline on mobile device
  
  - i.e. OpenCV for Android

- Some limitations due to power consumption

On-board pre-processing or even full processing

Ex. Tanskanen et al. Live Metric 3D Reconstruction on Mobile Phones. ICCV2013
Features (6/7): Connectivity

- **Many connectivity options**
  - Local area: NFC, Bluetooth, Bluetooth Low Energy, Wi-Fi 802.11x
  - Wide area: Cellular wireless networks: 3G/4G/5G

- **Mobile devices can connect at local or wide area at reasonable speed**
  - Typical LTE/4G: 18 Mbps down, 9.0 Mbps up
  - Typical Wi-Fi: 54Mbps (g), 300Mbps (n), 1Gbps (ac).

- **Lo-cost -> No-Costs**
**Features (6/7): Connectivity**

- Many connectivity options:
  - Local area: NFC, Bluetooth, Bluetooth Low Energy, Wi-Fi 802.11x
  - Wide area: Cellular wireless networks: 3G/4G/5G

- Mobile devices can connect at local or wide area at reasonable speed:
  - Typical LTE/4G: 18 Mbps down, 9.0 Mbps up
  - Typical Wi-Fi: 54Mbps (g), 300Mbps (n), 1Gbps (ac).

- Lo-cost -> No-Costs

---

**Load balancing (client / server)**

**Access to large databases (e.g., search)**

**Communication**

Features (7/7): Display!

- **Hi-res/hi-density display**
  - Data presentation!
  - Large with respect to processing power

- **Co-located with camera + other sensors**
  - Tracking during capture!

- **Touch screen**
  - Co-located user-interface
  - Small with respect to fingers (precision, occlusions!)
  - (UI also may exploit other sensors)

![Graph showing display resolution trends from 2011 to 2017](image)

*Data source: NPD DisplaySearch*
Features (7/7): Display!

- **Hi-res/hi-density display**
  - Data presentation
  - Large with respect to processing power

- **Co-located with other sensors**
  - Tracking during capture

- **Touch screen**
  - Co-located user-interface
  - Small with respect to fingers (precision, occlusions!)
  - (UI also may exploit other sensors)

---

**Data/result presentation**

**Guided capture / Augmentation**


![Graph showing display resolutions over years: 3840(UHD), 2460(WQHD), 1920(FHD), 1280(HD), <HD. Data source: NPD DisplaySearch*](image)
Wrap-up: mobile apps characterized by the exploitation of mobile device features

- **Features**
  1. Mobility
  2. Camera
  3. Active light
  4. Non-visual sensors
  5. Processing power
  6. Connectivity
  7. Display

Next session: case studies!
Part 5.2

Mobile Metric Capture & Reconstruction:
Case studies in metric capture

Giovanni Pintore, CRS4
Example 1

DATA FUSION FOR METRIC CAPTURE
Metric acquisition with a commodity mobile phone

- **Goal**
  - Capture 3D models with real-world measures

- **Data fusion approach**
  - Exploit synchronization of visual sensor & IMU to capture scenes in real-world units

Garro et al. *Fast Metric Acquisition with Mobile Devices*. VMV 2016
Structure-from-Motion + Dense reconstruction

• SfM reconstructs a point cloud from a series of images
  – 3D positions of (sparse) matched features
  – Camera positions and orientations

• Many approaches for densification
  – Pipeline showed to work at interactive rates on phones (Taskanen et al 2013)

• SCALE AMBIGUITY
Data fusion: Visual + IMU

• Use sensors synced with visual channel
  – GPS+Magnetometer generally not applicable
  – IMU returns orientation and acceleration in real world units

• Idea
  – track camera movement with IMU during visual capture
  – use IMU data to find out the real-world distance between SfM camera positions, resolving the scale ambiguity
Data fusion: Visual + IMU

- The accelerometer returns acceleration
- Therefore, we should be able to compute the displacement between two camera positions as

\[ x(T1, T2) = \left\| \int_{T1}^{T2} \left( v(T1) + \int_{T1}^{t'} a(t) \, dt \right) \, dt' \right\| \]

- Not so easy: onboard IMU sensors are biased and noisy and SfM camera positions are sparse
Data fusion approaches (1/4)

- Match position from IMU integration with position from SfM, coping with noise/bias by extensive filtering

  A new approach to vision-aided inertial navigation [Tardif et al 2010]

- Requires LONG acquisition times and LONG offline processing times

Data fusion approaches (2/4)

- **Ad-hoc online solutions taking into account IMU characteristics**
  - Segment motion in “swift movements” with large accelerations
  - Integration of IMU acceleration to derive position matched with SfM
  - Continuous process of outlier rejection and re-estimation of scale

- **Working but motion-dependent and prone to accumulation error due to integration**

Live metric 3D reconstruction on Mobile Phones [Tanskanen et al. 2013]

\[
\arg \min_{\lambda} \sum_{i \in I} \| \vec{x}_i - \lambda \vec{y}_i \|^2
\]

One estimate of \( \lambda \) at the end of each swift movement
Estimation of scale \( \lambda \) only on inlier set \( I \)
Data fusion approaches (3/4)

- Match accelerations from IMU with accelerations from SfM at SfM frame-rate (large baseline!)
  - **Downsample and anti-alias** IMU samples at SfM frame rate
  - Optimize scale and bias

Hand-waving away scale [Ham et al. 2014]

\[
\arg\min_{s, b} \eta \{ s \cdot \hat{A}_V + 1 \otimes b^T - DA_I R_I \}
\]

**D:** convolutional matrix for antialiasing and downsampling IMU signal

- Requires very long acquisition times due to downsampling at SfM rate
Data fusion approaches (4/4)

• Match accelerations from IMU with accelerations from SfM at IMU frame-rate (small baseline!)
  – **Upsample** SfM samples at high rate using all available visual data
  – Estimate acceleration from upsampled transforms and match them to IMU samples using robust fitting

Fast Metric Acquisition with Mobile Devices. [Garro et al. 2016]

\[
\text{argmin}_{s,R}\{\|A_c - sRA_s\|}\]

• Fast, coping with large errors and noise
Vision Module Pipeline
Vision Module

- Traces Shi-Thomasi features
- When baseline is large enough
  - Estimate Essential Matrix, that is, relative camera pose between f0 and fi
  - Calculate a 3D point for each feature point
- Note: each pair of cameras has its own reference system
Vision Module

- **Global registration**
  - M point clouds
  - A subset of features is present in each point cloud
  - Use feature correspondence to align all the point cloud in the same reference system

- **Cameras upsampling**
  - Features are tracked for **all** frames
  - Use aligned point cloud and tracking position to estimate cameras for all frames with Perspective-n-Point (PnP)
Recovering the scale factor (1/2)

**IMU accelerations**

\[
A_s = \begin{pmatrix}
    a_s^x(t_0) & a_s^y(t_0) & a_s^z(t_0) \\
    \vdots & \ddots & \vdots \\
    a_s^x(t_K) & a_s^y(t_K) & a_s^z(t_K)
\end{pmatrix}
\]

**Camera accelerations**

\[
A_c = \begin{pmatrix}
    p_c''(t_0)^T R_c(t_0) \\
    \vdots \\
    p_c''(t_K)^T R_c(t_K)
\end{pmatrix}
\]

**Problem to solve**

\[
\arg\min_s \{ \| A_c - s A_s \| \}
\]
Recovering the scale factor (2/2)

- LS, gradient descent (et similia) poorly conditioned
  - Not so many data
  - Severe outliers

- Robust fitting use RANSAC approach
  - Use MLESAC robust estimator to maximize likelihood rather than just the number of inliers

- Introduce rotation matrix $R$
  - Account for orientation bias
  - Improve RANSAC performance

\[
\begin{align*}
\arg\min_s \|A_c - sA_s\| \\
\end{align*}
\]

\[
\begin{align*}
\arg\min_{s,R} \|A_c - sRA_s\| \\
\end{align*}
\]
## Results

<table>
<thead>
<tr>
<th>Scene Name</th>
<th>Real scale m / s.u.</th>
<th>Acquisition info</th>
<th>Our approach m / s.u.</th>
<th>Simple scaling m / s.u.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer</td>
<td>2.094</td>
<td>17.0 65 883</td>
<td>2.01 4.0%</td>
<td>2.85 36.1%</td>
</tr>
<tr>
<td>Scanner setup</td>
<td>3.565</td>
<td>9.8 53 641</td>
<td>3.45 3.1%</td>
<td>3.12 12.4%</td>
</tr>
<tr>
<td>Desktop</td>
<td>6.520</td>
<td>11.3 48 596</td>
<td>6.24 4.2%</td>
<td>5.16 20.8%</td>
</tr>
<tr>
<td>Statuettes</td>
<td>2.602</td>
<td>11.5 53 607</td>
<td>2.49 4.5%</td>
<td>2.48 4.9%</td>
</tr>
<tr>
<td>Office desk</td>
<td>1.977</td>
<td>30.4 88 471</td>
<td>2.01 1.8%</td>
<td>2.01 1.8%</td>
</tr>
<tr>
<td>Office workstation</td>
<td>3.95</td>
<td>12.3 37 1307</td>
<td>3.94 0.3%</td>
<td>3.98 0.6%</td>
</tr>
<tr>
<td>Ara pacis</td>
<td>1.568</td>
<td>30.07 77 1569</td>
<td>1.52 2.8%</td>
<td>1.80 13.0%</td>
</tr>
<tr>
<td>Workstation (Fastest)</td>
<td>0.707</td>
<td>9.9 34 1305</td>
<td>0.73 2.7%</td>
<td>0.89 20.4%</td>
</tr>
<tr>
<td>Desk fast motion</td>
<td>6.918</td>
<td>14.8 74 1718</td>
<td>6.28 9.1%</td>
<td>3.88 44.0%</td>
</tr>
</tbody>
</table>

- Median error 4% (wrt 10-15% of other STAR solutions)
Example 2

DATA FUSION AND COMMUNICATION FOR INDOOR CAPTURE
Indoor capture + presentation

• **Creation and sharing of indoor digital mock-ups**
  - Exploiting the capabilities of modern mobile devices

• **Much interest/applications (security, location awareness, …)**
  - Need to capture visual information together with room structure
Typical solutions

- **Indoor capture and modeling**
  - Manual modeling
  - Semi-automatic methods based on high-density data
    - Laser scanning
      - Professional but expensive, limited to specific applications
    - Multi-view stereo from photographs
      - Generally cost effective but hard to apply in the indoor environment
        » Walls poorly textured, occlusions, clutter
        » Furthermore: need for heavy MW constraints, computationally demanding
Examples using low-cost mobile devices

- **Interactive capture and mapping of indoor environment**
    - Floor corners marked via an augmented reality interface
    - Manual editing of the room and floor plan merging using the screen interface
  - Sankar and Seitz: Capturing indoor scenes with smartphones (UIST2012)
    - Corners marked on the screen during video playback

Sankar et al. *Capturing indoor scenes with smartphones*, UIST2012
Exploiting panoramic images

• 360 degrees images are easy to capture using common devices
  – Interactive apps using IMU + GUI + automatic stitching
  – Dedicated cameras

• 360 degrees images are easy to navigate
  – Spheremaps + emerging formats
  – VR devices for immersion

• What about analyzing them?
Finding the room structure

- **Take one spheremap per room**
  - Equirectangular images generated by a mobile device
    - Vertical lines aligned with the gravity vector
    - Image approx. oriented towards magnetic North
  - Eventually use IMU + Visual features for stitching

- **Track user motion to identify connections between rooms**
  - Use IMU + Visual Features for tracking

- **Solve local + global optimization to find indoor structure**
  - Multi-room environment
Finding the room structure

- **Analyze spheremap to extract single room structure**
  - Room model considers vertical walls
  - Extract edges and filter out regions likely far from top/bottom edges of walls
  - Find wall height
    - Voting scheme used to extract most likely wall height by maximizing pairs of matching wall-floor / wall-height edge pixels
  - Fit 2.5D room model to recovered wall edge map

- **Uses specialized transform to speed-up computation**
Finding the rooms structure

- **Iterated to map the entire floor-plan**
  - Mobile tracking of user’s direction moving between adjacent rooms creates a connected room graph
  - Doors position identification in the image by computer vision
  - Doors matching according with graph
  - Rooms displacement
  - **Global optimization of combined model**

Pintore et al. *Omnidirectional image capture on mobile devices for fast automatic generation of 2.5D indoor maps*. IEEE WACV 2016
Results

<table>
<thead>
<tr>
<th>Scene Name</th>
<th>Features</th>
<th>Area error</th>
<th>Wall length error</th>
<th>Wall height error</th>
<th>Corner angle error</th>
<th>Editing time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area [m²] Np</td>
<td>MP Ours</td>
<td>MP Ours</td>
<td>MP Ours</td>
<td>MP Ours</td>
<td>MP Ours</td>
</tr>
<tr>
<td>Office H1</td>
<td>720 10</td>
<td>2.95% 1.78%</td>
<td>35 cm 15 cm</td>
<td>2.0 cm 1.2 cm</td>
<td>0.8 deg 0.8 deg</td>
<td>26m32s</td>
</tr>
<tr>
<td>Building B2</td>
<td>875 25</td>
<td>2.50% 1.54%</td>
<td>30 cm 7 cm</td>
<td>6.0 cm 1.5 cm</td>
<td>1.5 deg 1.5 deg</td>
<td>42m18s</td>
</tr>
<tr>
<td>Commercial</td>
<td>220 6</td>
<td>2.30% 1.82%</td>
<td>25 cm 8 cm</td>
<td>12.0 cm 2.7 cm</td>
<td>1.5 deg 1.0 deg</td>
<td>28m05s</td>
</tr>
<tr>
<td>Palace</td>
<td>183 3</td>
<td>16.86% 0.20%</td>
<td>94 cm 5 cm</td>
<td>45.0 cm 1.3 cm</td>
<td>1.8 deg 0.5 deg</td>
<td>15m08s</td>
</tr>
<tr>
<td>House 1</td>
<td>55 5</td>
<td>21.48% 2.10%</td>
<td>120 cm 16 cm</td>
<td>15.0 cm 4.7 cm</td>
<td>13.7 deg 1.2 deg</td>
<td>25m48s</td>
</tr>
<tr>
<td>House 2</td>
<td>64 7</td>
<td>28.05% 1.67%</td>
<td>85 cm 8 cm</td>
<td>18.0 cm 3.5 cm</td>
<td>15.0 deg 0.5 deg</td>
<td>32m25s</td>
</tr>
<tr>
<td>House 3</td>
<td>170 8</td>
<td>25.10% 2.06%</td>
<td>115 cm 15 cm</td>
<td>20.0 cm 4.0 cm</td>
<td>18.0 deg 1.5 deg</td>
<td>29m12s</td>
</tr>
</tbody>
</table>

Pintore et al. Omnidirectional image capture on mobile devices for fast automatic generation of 2.5D indoor maps. IEEE WACV 2016

- Reasonable, fast reconstruction with rough structure and visual features
Sharing the indoor model

**Indoor model**
- **Exploration graph**
  - Each node is a spheremap/room
  - edges (yellow) are transitions between adjacent rooms
  - Stored on a server (standard http Apache2)
- **Panoramic images**
  - Mapped according with the graph

**Interactive exploration**
- **Room**
  - **WebGL fragment shader**
  - dragging to change view orientation and pinching to zoom in/out
- **Passages**
  - **Real-time rendering** of the transitions between rooms
    - Exploiting geometric model stored on the server
    - Performance improvement compared to use precomputed videos
  - Suggested paths
Some results

Live demo: [http://vcg.isti.cnr.it/vasco/](http://vcg.isti.cnr.it/vasco/)
Click on the dataset on the left column to start

3D reconstruction of a 655 mq office with 19 rooms. This environment was acquired with a mobile phone (HTC One M8)

Reconstruction of a 70 rooms floor of the NHV ministry at Den Haag, Netherlands. The whole model was acquired with a Ricoh Theta S camera
Next session:

CLOSING/Q&A
Part 6

All good things come to an end…

(Bad ones, too)
Subject: Mobile Graphics

• All you need to know to get an introduction to the field of mobile graphics:
  – Scope and definition of “mobile graphics”
  – Brief overview of current trends in terms of available hardware architectures and research apps built of top of them
  – Quick overview of development environments
  – Rendering, with focus on rendering massive/complex surface and volume models
  – Capture, with focus on data fusion techniques
Contacts (in alphabetical order)

• Marco Agus (1,2)
  – Research Engineer at KAUST (Saudi Arabia)
  – Researcher at CRS4 (Italy)

• Enrico Gobbetti (1) - organizer
  – Director of Visual Computing at CRS4 (Italy)

• Fabio Marton (1)
  – Researcher at CRS4

• Giovanni Pintore (1)
  – Researcher at CRS4

• Pere-Pau Vázquez (3)
  – Professor at UPC, Spain

(1) www.crs4.it/vic/
(2) https://vcc.kaust.edu.sa
(3) http://www.virvig.eu/
Funding...

Center for Research, Development, and Advanced Studies in Sardinia, Italy

King Abdullah University of Science & Technology, Saudi Arabia

Polytechnic University of Catalonia, Spain

Project TDM
RAS - POR FESR 2014-2020

REGIONE AUTÓNOMA DE SARDIGNA
REGIONE AUTONOMA DELLA SARDEGNA
Projects VIGEC / VIDEOLAB

Spanish MINECO Ministry FEDER funds
Grant No. TIN2014-52211-C2-1-R
Thanks for your attention!

Q&A NOW (TIME PERMITTING...
More information...

Center for Research, Development, and Advanced Studies in Sardinia, Italy
www.crs4.it/vic/

King Abdullah University of Science & Technology, Saudi Arabia
vcc.kaust.edu.sa

Polytechnic University of Catalonia, Spain
www.virvig.eu