BSc (Hons) in Computing Science
Proposals for Student Projects

April 2016
# Contents

## Part A  Preapproved Proposals  

<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimising Path Tracing with Genetic Algorithms</td>
<td>Keith Bugeja, Joshua Ellul, Kevin Vella</td>
<td>7</td>
</tr>
<tr>
<td>Automating Pointcloud Acquisition methods for Scene Understanding</td>
<td>Sandro Spina, Keith Bugeja</td>
<td>9</td>
</tr>
<tr>
<td>A Generic Chess-Like Game Player</td>
<td>Gordon Pace, Jean-Paul Ebejer</td>
<td>10</td>
</tr>
<tr>
<td>Expressing Runtime Verification Properties Using Regular Expression in Larva</td>
<td>Gordon Pace, Christian Colombo</td>
<td>12</td>
</tr>
<tr>
<td>A Runtime Verification Tool for Javascript Programs</td>
<td>Gordon Pace, Christian Colombo</td>
<td>14</td>
</tr>
<tr>
<td>Expressing Versatile Runtime Verification Events for Java</td>
<td>Gordon Pace, Christian Colombo</td>
<td>16</td>
</tr>
<tr>
<td>Preliminary investigations on Runtime Enforcement Implementations</td>
<td>Adrian Francalanza</td>
<td>18</td>
</tr>
<tr>
<td>Proof Rule based Runtime Verification for LTL</td>
<td>Adrian Francalanza</td>
<td>20</td>
</tr>
<tr>
<td>Reusable Building Blocks for Thread and Event Scheduling</td>
<td>Kevin Vella, Keith Bugeja, Joshua Ellul</td>
<td>21</td>
</tr>
<tr>
<td>Language-Independent Native Address Spaces for the MMU-less</td>
<td>Joshua Ellul</td>
<td>23</td>
</tr>
<tr>
<td>Automatic Ahead-of-Time Compiler Generation</td>
<td>Joshua Ellul</td>
<td>25</td>
</tr>
<tr>
<td>A Dalvik Bytecode Interpreter for 16-bit Architectures</td>
<td>Joshua Ellul</td>
<td>27</td>
</tr>
<tr>
<td>A JavaScript Operating System Simulator</td>
<td>Joshua Ellul, Keith Bugeja, Kevin Vella</td>
<td>29</td>
</tr>
<tr>
<td>A Hybrid Smartphone-Cloud based Rendering Algorithm for Natural Video Scenes on Mobile Devices</td>
<td>Carl James Debono</td>
<td>30</td>
</tr>
<tr>
<td>Heart Beat Rate Calculation from Facial Video Recording on Smart-phones</td>
<td>Carl James Debono</td>
<td>31</td>
</tr>
</tbody>
</table>
Fast Video Compression
  Carl James Debono ........................................ 32
Multi-hop Data Broadcasting in Vehicular Networks
  Carl James Debono ........................................ 33
Super-resolution of License Plates
  Reuben Farrugia .......................................... 34
Emotion Analysis using Local Binary Pattern
  Reuben Farrugia .......................................... 36
Exploring Ligand-Protein Interactions for Drug Discovery
  Jean-Paul Ebejer ........................................ 37
The Malta Human Genome Project
  Jean-Paul Ebejer ........................................ 39
Motion Analysis with High Speed Video
  Johann A. Briffa ........................................ 40
Real-time pedestrian detection from a static video surveillance camera
  George Azzopardi ........................................ 41

Part B  Generic Proposals  43
  Using virtual machine introspection for kernel code execution monitoring
    Mark Vella, Kevin Vella ............................... 45
Extending Android’s Binder as a basis for application monitoring.
    Mark Vella, Joshua Ellul .............................. 47
Real-time radiosity for dynamic scenes
    Keith Bugeja, Sandro Spina, Kevin Vella ............ 49
Efficient rendering of shadowcasting light sources
    Keith Bugeja, Sandro Spina ............................ 51
Simulating formally specified language behaviour and analysis
    Adrian Francalanza .................................... 53
Algorithm Implementation on Highly Parallel Architectures
    Johann A. Briffa .................................... 54
Updates to Distributed Simulator for Communications Systems
    Johann A. Briffa .................................... 55
Machine learning methods for handling missing data in medical questionnaires
    Lalit Garg ............................................. 56
Real-time hospital bed occupancy and requirements forecasting
    Lalit Garg ............................................. 58
Supervisor Index ............................................ 60
Keyword Index ............................................... 62
Part A

Preapproved Proposals
This part of the booklet contains project proposals that have been preapproved by the Computing Science Board of Studies. Students who wish to pursue their Final Year Projects (FYPs) in a topic proposed herein are required to register their interest using the apposite form\textsuperscript{1} available online.

\textsuperscript{1}http://www.um.edu.mt/ict/UG/DayDegrees/fyps/FYPSCS/Registration
Optimising Path Tracing with Genetic Algorithms

Supervised by: Keith Bugeja, Joshua Ellul, Kevin Vella
Keywords: Computer Graphics, Ray Tracing, Genetic Algorithms

The path tracing algorithm [2] is a Monte Carlo solution to the rendering equation based on point-sampling methods. Recently, the algorithm has been the focus of a number of studies in interactive physically-based rendering due to its ability to simulate lighting phenomena such as global illumination and caustics. Being a Monte Carlo method, path tracing can quickly yield a coarse solution or, given more time, converge towards a more accurate one. Realtime applications opt for the former approach and have to contend with implementations that are largely optimised for a specific platform and hardware configuration, oft focusing on minimising execution time without factoring in power consumption.

While operation counts are correlated to performance, the increasingly complex hardware and compiler technology makes optimisation largely an empirical process [3]. The lack of semantic information available to conventional compilers limits their transformative power and the extent of the optimisations they can carry out. Thus, generating high-quality code still requires some form of human intervention. Library generators such as ATLAS [5] and FFTW [1] make use of semantic information to apply transformations at all levels of abstraction, and the most powerful of these are not just program optimisers but true algorithm design systems. ATLAS generates linear algebra routines, focusing the implementation on matrix-matrix multiplication, identifying the parameters that give the best performance on a given setup during installation. FFTW generates signal processing libraries using heuristics such as dynamic programming or genetic algorithms in the generation due to the search space being too large for exhaustive search to be possible.

The aim of this project is that of using genetic algorithms to improve program performance in a spirit similar to that shown effective for sorting and generator libraries like ATLAS and FFTW [4]. The candidate will identify the possible levels of abstraction at which transformations can be gainfully applied by an optimisation strategy and also determine the possible transformation units at each of these levels. A suitable evolutionary algorithm will be designed and developed to discover the transformations that best satisfy the desired optimisation criteria (e.g. lower execution time, lower power consumption or CPU utilisation).
Bibliography


Automating Pointcloud Acquisition methods for Scene Understanding

Supervised by: Sandro Spina, Keith Bugeja
Keywords: Computer Graphics, Pointclouds, Ray Tracing, GPU

While it may be a trivial task for a person to recognise objects and structures in an image or point cloud, this is not a straightforward task for a computer system. This task, referred to as scene understanding (SU), is particularly challenging in scenes comprised of an unknown number of different objects, where the complexity of the identification task is augmented with the added challenge of determining object boundaries. In cluttered indoor scenes objects are typically only partially visible to the sensor due to inter-object occlusions. In the general case, the utilisation of both appearance and shape information increases the potential for a correct interpretation of a scene.

A variety of SU from pointclouds algorithms exist, for instance Spina et al.[1] and Wang et al.[2], which propose graph-based methods for the identification of objects within cluttered indoor scenes. In both cases, indoor scenes are manually scanned and labelled in order to provide ground truth for evaluation purposes. This project looks into the development of a simulator framework to facilitate comparative evaluation of different SU algorithms on common datasets. This project will focus on the development of these datasets using either of the following approaches:

1) a manual (using triangulation-based scanners) approach. Main challenge is the design/implementation of a GUI to manually (or using semi-automated methods) label the manually acquired datasets. 2) a fully automated virtual scene generation approach. Main challenge is the design/implementation of a virtual scanner which properly simulates a manual acquisition process and automatically navigates the virtual scene to produce the pointcloud dataset.

Bibliography

[1] Sandro Spina, Kurt Deobattista, Keith Bugeja, and Alan Chalmers. Scene segmentation and understanding for context-free point clouds. 2014.

A Generic Chess-Like Game Player

Supervised by: Gordon Pace, Jean-Paul Ebejer
Keywords: Game Playing

Computer board game playing has hit the news again recently when a computer Go player beat top human players, but has a long history starting from the dream of chess playing automata like the mechanical Turk [6], and with computer playing algorithms initially discussed in the fifties [5, 4].

An interesting problem is that of analysing a class of games, as opposed to a single one. In this project, the student will be exploring the class of symmetric chess-like games, in which two players battle across a board upon which pieces are placed. Such games may vary with respect to the topology of the board, how the pieces move and capture, how turns work, etc. The project will build on the work of Pell [2] to build a system which allows a user to describe a game and automatically produce a client to enable playing according the given rules, and also a computer player for the game.

The project will be organised into phases as follows:

Research and understanding: The student will be expected to read through the papers in the project’s bibliography and identify other work which might be relevant or helpful to understand the algorithms and data structures involved.

Generic game framework: A framework to describe chess-like games, using a language akin to the one used in [2] will be built, enabling automated generation of clients to allow human players to play against each other such games.

Extending game information: The language will be extended to enable the automated analysis of a game position. This will be used to be able to generate computer players using standard algorithms such as minimax and alpha-beta pruning [1].

Extending the class of games: The student will investigate how the class of games can be extended beyond that used in [2] and show how this can be achieved.

Evaluation: The framework developed will be evaluated on the basis of (i) coverage of chess-like games, by showing whether and how easily instances of games from standard texts like [3, 2] can be encoded in the framework; and (ii) limitations to quality of automated play, by comparing the level of play achieved by the generic player to specialised game-specific players.
Bibliography


Expressing Runtime Verification Properties Using Regular Expression in Larva

**Supervised by:** Gordon Pace, Christian Colombo  
**Keywords:** Runtime Verification

Runtime verification [3] provides an approach to monitoring the behaviour of a system without directly changing the code, thus separating the concerns of normal system behaviour from that of the verification. Starting from a formal specification and a system, a runtime verification tool automatically instruments checks for the properties into the system, ensuring that any violations are identified at runtime.

Runtime verification tools come with their own native property language and although one can typically translate other languages into such the native one, relating errors back to the original property can be challenging. In this project, we will be using the runtime verification tool Larva [2], which uses an automaton-based formalism DATEs to describe properties, and build different ways of allowing properties to be written using regular expressions. Regular expressions have been used in different ways to express properties: (i) positive specifications which state how the system should behave under normal circumstances e.g. the property \( \text{login} (\text{read} + \text{write})^* \text{logout}^* \) shows that reading a writing should only occur while logged in; (ii) negative specifications which state what should not happen e.g. the property \( (\text{login} + \text{logout})^* (\text{read} + \text{write}) \) states that a read or a write while logged out is wrong behaviour. Although equivalent in terms of expressiveness, some properties may be easier to express in one of the two forms. Although translating regular expressions into DATEs can use standard algorithms, relating unexpected behaviour with the original regular expression specification can be challenging. For instance, receiving two consecutive logins would trigger an error in the first property given above, and would have to be explained in terms of the expression by showing that after a login, only read, write or logout should be allowed. The aim of this project is to explore different ways of achieving this.

The project will be organised into phases as follows:

**Research and understanding:** The student will be expected to read through the papers in the project’s bibliography and identify other work which might be relevant or helpful to understand the tools, algorithms and data.
structures involved.

Case study: An appropriate system will be identified — an open source Java project — for which the student will write a number of properties expressed using regular expressions, and implemented directly using DATEs in Larva.

Regular expression monitoring (I): A tool to translate a regular expression (positive or negative) into DATEs using standard techniques from formal languages [5] and using deterministic finite state automata will be built. The translation should ensure that violations of a property will be explained and logged in terms of the original regular expression rather than the DATE produced.

Regular expression monitoring (II): The translation to deterministic automata may lead to an exponential blowup. One way of circumventing this problem is that of producing non-deterministic automata and keeping track of the set of states the automaton may lie in using DATE variables. This approach will be implemented, including feedback using the original regular expression as in the previous case.

Regular expression monitoring (III): Another approach which has been proposed to monitor regular expressions is that of using derivatives [1, 4]. This approach will also be implemented as in the previous two cases.

Evaluation: The different approaches developed will be evaluated on the basis of (i) efficiency in terms of memory and temporal overheads induced; and (ii) complexity of reverse explanation of identified property violations in terms of the original regular expression.

Bibliography


A Runtime Verification Tool for Javascript Programs

Supervised by: Gordon Pace, Christian Colombo
Keywords: Runtime Verification

Runtime verification [3] provides an approach to monitoring the behaviour of a system without directly changing the code thus separating the concerns of normal system behaviour from that of the verification. Starting from a formal specification and a system, a runtime verification tool automatically instruments checks for the properties into the system, ensuring that any violations are identified at runtime.

Javascript is becoming an increasingly popular technology, and its widespread use to interact with web content results in an interaction of technologies which poses particular challenges for runtime verification.

The aim of this project is to build a runtime verification tool for Javascript programs, extended to enable the monitoring of events consisting not only of Javascript control-flow points (e.g. the moment a function is called, or exits) but also upon changes to the Document Object Model (DOM) with which the Javascript code is interacting. The tool will use a guarded-command language to specify properties in the same style as used by the runtime verification tool pollyRV [1].

The project will be organised into phases as follows:

Research and understanding: The student will be expected to read through the papers in the project’s bibliography and identify other work which might be relevant or helpful to understand the tools, algorithms and data structures involved.

Case study: An appropriate system will be identified — an open source Javascript project — for which the student will write a number of properties expressed using guarded-commands in the style of [1].

Aspect-Oriented Programming for Javascript: Typically, runtime verification tools use aspect-oriented programming (AoP) [2] techniques to inject code into a system to enable verification without having to change the code of the system directly. There are a number of AoP tools for Javascript, such as Meld meld\(^2\) and AspectJS\(^3\), which the student is to

\(^2\)https://github.com/cujojs/meld
\(^3\)http://www.aspectjs.com/
look into and select an appropriate one for building the runtime verification tool upon.

**Runtime verification for Javascript:** The student will build a tool for the monitoring of Javascript programs using standard AoP techniques.

**DOM modification events:** The set of events which the tool can handle will be extended to deal with modification of the DOM structure. Different approaches in which this can be done will be explored, but the tool may assume that access to the DOM will be done via jQuery\(^4\) or a similar library.

**Evaluation:** The tool will be developed will be evaluated on the basis of efficiency in terms of memory and temporal overheads induced in the case study. Other properties which can be scaled up (e.g. monitoring each instance of a DOM object, thus allowing for scaling up analysis to large documents) will also be analysed to give a better picture of the effectiveness of the approach.

**Bibliography**


\(^4\)https://jquery.com/
Expressing Versatile Runtime Verification Events for Java

Supervised by: Gordon Pace, Christian Colombo
Keywords: Runtime Verification

Runtime verification [3] provides an approach to monitoring the behaviour of a system without directly changing the code thus separating the concerns of normal system behaviour from that of the verification. Starting from a formal specification and a system, a runtime verification tool automatically instruments checks for the properties into the system, ensuring that any violations are identified at runtime.

System behaviour is typically defined in terms of a sequence of system events where an event is a point of interest in the system’s execution. To enable the specification of points of interest, several runtime verification tools come with their own native event specification language. While one would usually be able to express a wide range of events and bind a number of related objects (such as parameters and returned objects), such an events specification language might sometimes limit the expressiveness of the runtime verification tool.

In this project, we will be investigating the limitations of the event language used in the runtime verification tool Larva [1]. The Larva event language is closely related to AspectJ [2] but is a strict subset of it. While Larva has been used to monitor a numerous Java systems, some concepts (such as filtering classes by package name) are not directly expressible in the event language. Furthermore, and perhaps more importantly, Larva has been evolving over the years and some features are not well integrated and documented. Examples include detecting constructor events, distinguishing between execution and call events, binding the “this” object, and so on. Therefore, the aim of this project would be to look at a range of Java program events one would want to monitor and redesign the language, making sure to identify a good trade-off of expressiveness versus clarity of the event language.

The project will be organised into phases as follows:

Research and understanding: The student will be expected to read through the papers in the project’s bibliography and identify other work which might be relevant or helpful to understand the tools, algorithms and data structures involved.

Experimenting with Larva and AspectJ: A deep understanding of Larva and AspectJ is a must. Therefore, the second phase involves carrying out
exercises and experimentation with these tools.

**Case study:** An appropriate system will be identified — an open source Java project — for which the student will identify a variety of potential events of interest.

**Event language design:** An event language is designed, taking into consideration the knowledge experience gathered from the previous phases.

**Integration with Larva:** The language will be implemented and integrated as part of the Larva runtime verification suite.

**Evaluation:** By implementing the case study using (i) plain Larva; and (ii) the extended event language, a qualitative analysis of the expressiveness and clarity of the Larva specification language extension can be carried out.

**Bibliography**


Preliminary investigations on Runtime Enforcement Implementations

Supervised by: Adrian Francalanza
Keywords: Runtime Analysis, Concurrency

DetectEr and AdapterEr form part of a tool suite for extending a concurrency system (written in Erlang) with runtime monitoring support. They automatically synthesise and instrument monitors from logical specifications that analyse and adapt the system under scrutiny while it is executing. The tool suite has acted both as prototype tool guiding the study of theories for monitoring semantics [3, 4] as well as vehicle for investigating practical issues relating to aspects such as asynchronous/synchronous monitoring and monitor modularisation [1, 2].

In this project, the candidate will conduct preliminary investigations for an implementation of a prototype told that generates enforcement monitors. Whereas detection monitors merely flag behavioural violations/satisfactions, and adaptation monitors attempt to change aspects of a running system after a violation/satisfaction is detected, enforcement monitors attempt to anticipate violations before they actually happen. They are thus capable to enforce the satisfaction of a correctness property. The candidate will start from an existing implementation of the tool suite and investigate extensions for enforcement monitor instumentations and enforcement monitor synthesis. Since the monitors will have their own thread of execution that is separate from that of the system being analysed, the candidate will be exposed to concurrent programming and issues associated with this paradigm such as race conditions.

Bibliography


Proof Rule based Runtime Verification for LTL

Supervised by: Adrian Francalanza
Keywords: Runtime Verification, Logic

Runtime Verification (RV) [2] is a lightweight technique that attempts to verify the correctness of a system by analysing its behaviour at runtime. In most cases, correctness is specified using some formal program logic which a precise semantics, from which monitors are then synthesised to execute along side and analyse the system under scrutiny.

Linear Temporal Logic [3] is a commonly used logic in the setting of RV because its semantics can be given in terms of execution traces. In [1] the authors define a proof system that is attuned to the restrictions of online monitoring (e.g., partial traces and incremental analysis) with the advantage that property satisfactions/violations can be backed up by a proof derivation. They also outline a procedure for extracting monitors from this proof system. The candidate will be expected to understand this formal system and carry out the resp., monitor synthesis from these proof rules.

Bibliography


Reusuable Building Blocks 
for Thread and Event Scheduling

Supervised by: Kevin Vella, Keith Bugeja, Joshua Ellul
Keywords: System Software, Multicores

This project will focus on identifying common building blocks that resurface in several implementations of thread and event schedulers [5, 4, 1, 2, 3], isolating these reusable primitives in a library, and quantifying the impact of this reorganisation on application performance.

While thread and event schedulers tend to be packaged as monolithic general-purpose libraries that are isolated from application-specific code, there are also use cases where application performance benefits significantly from hard-coding task or event scheduling in the application itself. An example of this can be found in parallel scene rendering, where it is common to bind the notion of a task to the specific application. This project will investigate an alternate method of structuring the system where a collection of lower-level scheduling primitives are exposed to the application. Empirical measurement will determine whether this compromise can yield comparable performance to hard-coded, application-specific scheduler implementations while maintaining the ease-of-use and generality of a scheduling library.

Work on this project will be organised in the following phases:

- Implement a compendium of generic and task-specific user-level schedulers for threads and events on single and multicore processors using a selection of locking and lock-free primitives.

- Identify and isolate common building blocks in a library and reimplement the schedulers using this library.

- Measure, compare and analyse the performance impact of using the library relative to the original implementations.

An good knowledge of C is required, and experience with x86 assembly language would be beneficial. Background knowledge on system software, operating systems and concurrent systems is desirable. In the course of the project the student will gain experience on system software, concurrent systems, lock-based and lock-free techniques, performance measurement and analysis.
Bibliography


Language-Independent
Native Address Spaces for the MMU-less

Supervised by: Joshua Ellul
Keywords: Operating Systems, Compilers, Systems

The Internet of Things (IoT) promises to enhance and augment our lives by enabling connected devices to monitor and react to our daily routines. It is envisaged that users will be able to download apps onto their different devices in aim of customising and getting the most out of the devices as possible. Downloading applications from untrusted (and even trusted) sources introduces the potential for malicious and/or buggy applications to disrupt behaviour of other applications. Traditional processors provide memory management units (MMUs) that enable operating systems to provide protection between address spaces of the different executing processes. Many microcontrollers used in the IoT domain do not have MMUs.

Wahbe et al [4] first proposed software-based techniques for process memory protection. More recently software-based approaches to enable process memory protection for such resource constrained devices have been proposed [1] [3] [5]. The techniques proposed are either language-dependent [3], require a virtualisation layer [5] or are no longer supported or available [1].

LLVM [2] is a modular and reusable compiler infrastructure that is source-language and target-architecture-independent. A compilation pass that inserts run-time memory-access checks can be implemented in LLVM so as to provide programming language-independent process memory protection that is executed natively. In this project we aim to investigate software-based techniques to enable process memory protection that are programming language-independent and natively supported.

Bibliography


Automatic Ahead-of-Time Compiler Generation

Supervised by: Joshua Ellul
Keywords: Operating Systems, Compilers, Systems

Intermediate representations (IR) are useful for code distribution due to their platform independence and smaller size. Upon receiving code it is often beneficial to translate received IR into the underlying native architecture instruction set so as to benefit from execution gains (as opposed to having to interpret the IR). This process is called Ahead-Of-Time (AOT) compilation. Writing an AOT compiler is a non-trivial task [3]. It involves mapping from a source language or instruction set to the underlying target native instruction set.

LLVM [5] is a modular and reusable compiler infrastructure that is source language and target architecture independent. New languages can be supported by writing a frontend for LLVM, and new target platforms can be supported by writing a backend for LLVM. LLVM has gained extensive interest from both academia and industry which has resulted in quite a number of frontends [1][2] and backends [4][6] being made available.

In this project we aim to investigate techniques to facilitate automatic generation of AOT compilers for different architectures by analysing how the different input IR instructions are translated to native instructions for different target architectures using available LLVM backends. Heuristics will then be proposed that given the input IR instructions and the generated native code instructions, will output an AOT compiler for the respective architecture.

Bibliography


A Dalvik Bytecode Interpreter for 16-bit Architectures

Supervised by: Joshua Ellul
Keywords: Operating Systems, Compilers, Systems

Over the past decade the Android Operating System has become ubiquitous within mobile platforms. Prior to the Lollipop Android 5.0 release Android’s execution engine, Dalvik, was based on Just-In-Time (JIT) compilation and interpretation of Dalvik bytecode. Thereafter a new execution engine, the Android Runtime (ART), was introduced which performs Ahead-Of-Time (AOT) compilation of Dalvik bytecode to the underlying native instruction set. Although the Dalvik execution engine has been replaced, Dalvik bytecode is still an integral part of Android and the runtime.

Android is also gaining traction within the Internet-of-Things (IoT), specifically on larger devices. The operating system’s footprint is much larger than what many resource constrained systems typically used in the IoT can support. However, the Dalvik bytecode instruction set and a subset of the Android framework could be used on such resource constrained systems. Dalvik bytecode is similar to Java bytecode except that it is a register-based instruction set as opposed to a stack-based one. Previous work has shown that interpretation [3] and dynamic binary translation [4] of Java bytecode are possible on such resource-constrained devices.

In this project we aim to investigate techniques to efficiently interpret Dalvik Bytecode on resource constrained 16-bit microcontrollers. We will aim to specifically implement an interpreter for MSP430 microcontrollers, however will also focus on facilitating other resource constrained microcontrollers. In doing so we will investigate intermediate representation design [1], garbage collection [2] and memory management [3] techniques.

Bibliography


A JavaScript Operating System Simulator

Supervised by: Joshua Ellul, Keith Bugeja, Kevin Vella

Keywords: Operating Systems, Systems

Operating System (OS) concepts are a foundation of many areas of computer science. Understanding what is under the hood can help even high-level programmers to develop efficient code. A practical approach to teaching operating systems by demonstrating internals is beneficial. However, available popular operating system source code is often too complex for the purpose of demonstrating internal concepts.

Instructional operating systems specifically designed for demonstration of OS concepts were first proposed in the early 1980’s [3] of which many were motivated by the development of UNIX [2]. Such operating systems required students to install the OS on a separate partition or VM, or required configuration that posed a barrier to entry. More recently, web-based OS simulators were proposed. However, they are either no longer publicly available [1] or require browsers that support non-standard frameworks such as silverlight [4].

In this project, a client-side JavaScript Operating System Simulator will be developed which will provide a skeleton framework of an Operating System that will allow users to plug-in their own implementation of Operating System level algorithms.

Bibliography


A Hybrid Smartphone-Cloud based Rendering Algorithm for Natural Video Scenes on Mobile Devices

Supervised by: Carl James Debono
Keywords: Video Processing, Cloud Computing, Computer Graphics

Nowadays, mobile devices are being equipped with more processing power and graphical processing units allowing them to perform more complex tasks. Free-viewpoint television is a solution that allows the user to select a view where this can be an actual camera view or a virtual position that has to be generated from the available views [1]. Rendering these virtual views at 25 or 30 frames per second or better for high-definition screens is very computationally demanding even for the latest devices and drains battery power fast. In this project, the student will reduce the mobile device’s workload by offloading part of these computations to the cloud. The developed algorithm will need to be tested and evaluated in terms of speed and video quality [2].

Bibliography


Heart Beat Rate Calculation from Facial Video Recording on Smartphones

Supervised by: Carl James Debono
Keywords: Video Processing, Digital Image Processing, Artificial Intelligence

Mobile devices have become portable computers which are equipped with cameras, colour displays, and other sensors. A non-invasive technique to measure heart beat is to use video processing [2]. Image processing tools are used to determine the location of the face of the user as the area of interest [1]. This is followed by video processing algorithm to identify and count the number of heart beats per minute. This can be displayed on the screen or transmitted as necessary. Given that the video is taken in real-life scenarios, motion induced artefacts and ambient noise have to be filtered. The aim of this project is to create a solution that is low in complexity, fast, and reliable with the resources available on a typical smartphone.

Bibliography


Fast Video Compression

**Supervised by:** Carl James Debono  
**Keywords:** Video Processing, Data Compression, Computer Graphics

The latest video compression standard (H.265 or HEVC) [4] is quite complex [1] due to the amount of predictions and searches required in optimising compression. This results in delays that make it unpractical in resource-restricted environments. Therefore there is scope to try to minimise time complexity by employing better prediction schemes [3] and parallelisation [2] of the algorithms used. The aim of this project is therefore to explore techniques and develop algorithms to minimise the time taken for compression. The study will include the impact of these algorithms on the quality of the video.

**Bibliography**


Multi-hop Data Broadcasting in Vehicular Networks

Supervised by: Carl James Debono
Keywords: Data Networks, Graph Theory, Artificial Intelligence

New vehicles are being equipped with on-board computers and soon will carry wireless devices that can transmit and receive data. Moreover, vehicles have a number of sensors ranging from telemetery measuring devices to cameras, which are interfaced to on-board stand-alone computer systems. The addition of wireless transmitting devices will allow communications with other vehicles and with the infrastructure and can help in improving road safety. Broadcasting is ideal for vehicular networks since fast dissemination of critical information is crucial for its effectiveness. However, the reliability of transmission cannot be guaranteed in such an environment. The aim of this project is to develop a multi-hop broadcasting protocol [2, 1] and investigate its performance compared to simple flooding of the network.

Bibliography


Super-resolution of License Plates

Supervised by: Reuben Farrugia

Keywords: Digital Image Processing, Image Restoration, Super-Resolution

Videos captured by Closed Circuit Television (CCTV) systems are important in many areas such as crime scene investigation and monitoring offences in public roads [4]. These cameras are normally installed to cover a large field of view where the query license plate may not be sampled densely enough by the camera sensors. The low-resolution of the captured images reduce the effectiveness of CCTV in identifying perpetrators and potential eyewitnesses [3].

The goal of super-resolution (SR) methods is to recover a high resolution image from one or more low resolution input images [2]. Class-specific example-based restoration methods have been successfully employed in face super-resolution [5], iris-super-resolution [1] and face-inpainting [6] to mention a few. In this project we will investigate the use of class-specific example-based super-resolution methods for license-plate super-resolution. The developed algorithm will extract the low-quality license plate image and perform class-specific example-based super-resolution to increase the resolution and texture detail in the restored image. Experiments will be conducted on publicly available datasets 5 6. The project can be developed using either C++ or Python.

Bibliography


License Plate Dataset: http://www.zemris.fer.hr/projects/LicensePlates/

MediaLab LPR Dataset: http://www.medialab.ntua.gr/research/LPRdatabase.html


Emotion Analysis using Local Binary Pattern

Supervised by: Reuben Farrugia
Keywords: Computer Vision

Affective computing, which is currently an active research area, aims at building the machines that recognize, express, model, communicate and respond to a user’s emotion information [2, 3]. Within this field, recognizing human emotion from facial images, i.e., facial expression recognition, is increasingly attracting attention and has become an important issue, since facial expression provides the most natural and immediate indication about a person’s emotions.

An automated facial expression recognition system generally comprises two steps: i) facial feature extraction and ii) facial expression classification. In this project the student will use Local Binary Pattern (LBP) as facial features to extract discriminative features to be used for classification. LBP is a texture descriptor which was used in several applications, including emotion analysis [4, 1]. The classification stage can be computed using linear multi-category classifiers using publicly available datasets 7 8. The project can be implemented using MATLAB, C++ or Python.

Bibliography


7 Emotion Dataset: http://mmspg.epfl.ch/emotion_dataset/
8 DEAPdataset: http://www.eecs.qmul.ac.uk/mmv/datasets/deap/
Exploring Ligand-Protein Interactions for Drug Discovery

Supervised by: Jean-Paul Ebejer
Keywords: Data Science, Bioinformatics, Cheminformatics

Small-molecule drugs work by binding to a protein to either inhibit its function or to activate it. The small-molecule, hereafter referred to as ligand, has a 3D structure which is complementary to the protein’s binding site. The ligand binds to the protein due to interactions between the ligand and the protein (e.g. a favourable interaction could feature a positive charge on the ligand and a negative charge on the protein within a certain distance). Even though many such interactions exist [2], and are well understood, calculating the free binding energy is still an elusive problem.

Figure 1: Aspirin (in yellow) bound to a protein in PDB structure 1TGM.

The Protein Database [1] is a repository of over 100,000 experimentally-resolved protein structures (with 3D coordinates). Some of these structures have bound ligands which interact with the protein. The aim of this project is to identify and mine these structures and build a database of protein-ligand interactions. The interaction type, ligand-to-protein distance, quality of the structure will be extracted. Many applications of such a database exist, including training models to predict binding affinity and to elucidate the determinants
of molecular recognition.

The main deliverables of this project are:

- An automated process to determine protein-ligand complexes of interest from the PDB (note that this also includes evaluating the “goodness” of the structure)
- An automated process to analyse these complexes and extract protein-ligand interactions
- Design and implementation of a protein-ligand interactions database (e.g. including querying capabilities for specific classes of protein-ligand interactions using molecular similarity)
- Statistical analysis of protein-ligand interactions
- A website advertising results and allowing the user to download the full dataset
- A RESTful API which allows users to query the database

This project is best suited to students with programming experience (Python, Java, etc.) with an interest in bioinformatics and biological systems.

Bibliography


The Malta Human Genome Project

Supervised by: Jean-Paul Ebejer
Keywords: Bioinformatics, Data Science, Big Data

The University of Malta shall be, for the first time in Malta, developing a National Maltese Human Reference Genome in the form of a public database and a user-friendly web-based viewer. The overall intention is to discover new DNA variants that cause disease in Malta and may be developed into biomarkers for diagnosis or targets for the development of new drugs.

Together with the partner help of Complete Genomics, which is an established human genome sequencing facility based in the USA, we shall be conducting whole genome sequencing on selected Maltese DNA samples available from the established Malta BioBank located on campus. What initially costed 2 billion euros and took 10 years to complete (from 1990 to 2000), the whole human genome can now be sequenced in less than a day and under 1000 EUR per genome.

The setting up of a National Maltese Human Reference Genome database will have lasting benefits for the Health Sector and major impact on the society. Clinicians from different sectors can compare their patients’ results with this reference genome and pinpoint exactly the underlying molecular lesion. The patient can benefit from personalized treatment and medicine making healthcare more efficient and cost-effective. The presence of this database will also in turn encourage clinicians to conduct, perform and order genetic tests to screen whole exomes/genomes of their patients, whereas previously they would have been reluctant if such a reference was unavailable.

The aim of this project is to build tools for the visualization and analysis of genomes sequenced from the Malta Human Genome Project. This is a ‘Big Data’ project (each individual’s DNA has 3 billion bases), which requires special techniques in data storage, visualization and analysis.

The main deliverables of this project are:

- Visualization of human genomes. This includes a comparative review of existing methods.

- Building of a genome browser using novel and established components which allow for comparative genomics (to reference DNA) and analysis of DNA mutations.

This project is best suited to students with programming experience (Python, Java, etc.) with an interest in bioinformatics and biological systems.
Motion Analysis with High Speed Video

Supervised by: Johann A. Briffa
Keywords: Digital Image Processing, Embedded Systems, Software Engineering

The project trials the use of recently available low cost high speed cameras in a sports application and develop the necessary software to enable that use. Similar systems have existed for some time, though at significantly higher costs (in the order of €10–100k per unit). Recent advances in sensor technology also means that cameras are now able to operate with less light, which has always been a limiting factor. While some open source software is already available, its functionality will not necessarily cover all the requirements for the intended application. Therefore, some programming effort will be required to implement the necessary updates. The chosen camera platform exposes the embedded microcontroller, allowing us to modify how the hardware operates, as needed.

Objectives:

• Consider off-the-shelf software for motion tracking (e.g. [1, 2, 3]) and apply this to target archery with the available high speed camera.

• Suggest improvements, listing upgrade features and performing a preliminary analysis of how these could be implemented.

• Design, implement, and test a selection of features identified above.

Student background/interests:

• Interest in image processing and/or embedded systems.

• Aptitude and willingness to program (the project requires the use of C++; prior OO development experience in another language is suitable).

Bibliography


Real-time pedestrian detection from a static video surveillance camera

Supervised by: George Azzopardi
Keywords: High Performance Computing, Computer Vision, Machine Learning, Digital Image Processing

The automatic detection of pedestrians is very important and has various applications, ranging from security on the roads, tracking of pedestrians in high security areas (e.g. airports, train stations), and detecting abnormal behaviour from the gait analysis of pedestrians. Fig. 1 shows an example of pedestrian detection system. There are various challenges in such applications, which mainly are due to various light conditions (day, night, sunny, cloudy) and occlusions.

In this project, we will investigate the effectiveness and efficiency of various algorithms which performs the detection of pedestrians in real-time. Techniques that will be investigated include the histogram of gradients descriptor (HOG) [2] as well as the COSFIRE approach [1]. For the evaluation of this project we will use public benchmark data sets as well as a video surveillance camera that we will install in a public space.

This project is ideal for students who are interested to develop efficient algorithms for real-time applications. No prior knowledge about computer vision techniques is required.

Figure 2: Example of pedestrian detection system; taken from [3]
Bibliography


Part B

Generic Proposals
This part of the booklet contains a series of generic proposals which have **not** been preapproved. Together with their supervisors, students may use any of these proposals as a starting point in crafting a specific submission to the Computing Science Board of Studies for approval. Therefore, students interested in basing their FYPs on one of the generic proposals presented herein should:

1. Discuss the proposal(s) with at least one of the listed academic members of staff.

2. Upon acceptance of supervision by the academic member of staff, complete the apposite acceptance form\(^1\) available online.

3. Await notification of approval by the Computing Science Board of Studies.

\(^1\)http://www.um.edu.mt/ict/UG/DayDegrees/fyps/FYPCS/Registration
Using virtual machine introspection for kernel code execution monitoring

Supervised by: Mark Vella, Kevin Vella
Keywords: Cyber Security, Digital Forensics

Memory analysis routines are currently used to conduct digital investigations [3, 5] and to automate malware analysis [4]. However the benefits of both these applications are currently limited by the off-line analysis of memory dumps. In the prior case memory dumps are typically only acquired at the start of an incident response procedure [8], typically on the flagging of a suspicious event. Whilst providing a valuable source of forensic artifacts in case of malicious activity, that either does not touch secondary storage or else encrypts its artifacts, the volatile nature of memory objects complicates matters when dumps are not acquired in a timely fashion. For the latter case, malware sandboxes typically create a snapshot of system memory before and after malware execution, with any differences being highlighted as a result of their comparison. This approach is particularly suitable for analyzing kernel-level rootkits that manipulate kernel-level structures to hide their presence along with any user-level components. Whilst benefiting the triaging of suspicious binaries this approach is a far cry from providing a detailed account of code execution. A finer-grained approach for automating dynamic malware analysis can pursue the Dynamic Binary Translation [2] approach, which however is tough to pull off at the kernel-level due to significant threats for performance and system stability.

This group of (individual) projects aims to explore how to leverage an existing bare-bones virtual machine introspection framework [9] in order to trigger memory analysis routines in a timely fashion as well as to produce finer-grained accounts of kernel-level malware behavior during automated analysis. Comparative evaluation with live forensics and off-line memory dump analysis routines will be used to measure technique effectiveness and performance costs, whilst case studies using real-world rootkits will demonstrate their value in a cyber security setting. The projects will cater for both Windows [7] and Linux-based [6] operating systems, covering the x86/x86-64 and ARMv7-A [1] platforms. At the undergraduate level, projects will only be expected to address core and document kernel structures (processes, files, and network sockets) and to only evaluate the effectiveness aspect. The number of students working on these projects is limited by hardware availability in the Systems Security Lab (unless provided otherwise).
Bibliography


Extending Android’s Binder as a basis for application monitoring.

Supervised by: Mark Vella, Joshua Ellul
Keywords: Cyber Security, Digital Forensics

Android is a Linux-based open-source operating system (OS) and is currently the market leader for smart devices\(^2\). By design this OS aims for a simplified and elegant application development framework, while at the same time fitting the constrained resources of smart devices it targets for deployment\(^3\). One result of this aim constitutes Android’s framework (Java-level) application component communication mechanisms of: Intents, Intent Filters, Bound Services and any associated Message Handlers \([3, 5]\), which provide application developers with a high-level of abstraction mechanisms to connect components within and across applications. Underneath, all these mechanisms are implemented using a central Remote Procedure Call (RPC) mechanism, named Binder \([1]\). Its implementation is split between Android’s middleware (user-level Linux/C++ implemented shared object) and the Linux kernel (a C-implemented driver). Binder replaces the Linux’s System V inter-process communication (IPC) mechanisms, with the aim of rendering this central component as efficient as possible and thus avoids introducing a serious performance bottleneck. Its centrality makes it also a convenient choke point for monitoring both application and system-centric behavior. Such a mechanism could benefit the development of verification and security tools alike \([2]\).

This group of (individual) projects aims to explore how to leverage Binder’s strategic positioning within Android’s architecture in order to provide application monitoring functionality, which can be tapped by application developers at the Android framework level while still using existing developer abstractions. This work requires extending Binder’s current implementation whilst safeguarding it from malicious subvertion by malware to leak sensitive information. In this respect it is necessary to patch the modify Android’s SELinux policy. SELinux is a complementary Discretionary Access Control mechanism for Linux-based OSs, that separates access control policy definition from enforcement \([4]\). Additionally, it would be interesting to explore the use of in-memory patching through Linux’s ptrace interface \([6]\) in order to bring the required Android modifications to a minimum. Specifically, most likely it would require just a change to the initializing process’s and SELinux’s configuration. Another

\(^{2}\)http://www.idc.com/prodserv/smartphone-os-market-share.jsp
\(^{3}\)https://source.android.com/
advantage of this approach could be the possibility to toggle the availability of additional monitoring through the simple insertion/removal of an SD card. This proposition mechanism requires evaluation in terms of: i) seamless blending with Android’s application development framework, ii) performance overheads, iii+iv) threats to system stability, and security. At the undergraduate level, the study is expected to only cover application events concerning framework-level system service usage, while at the postgraduate level it is expected to also cover full application component activation. Also, the undergraduate level excludes venturing into the security and in-memory patching aspects of this study.

Bibliography


Real-time radiosity for dynamic scenes

Supervised by: Keith Bugeja, Sandro Spina, Kevin Vella

Keywords: Computer Graphics, Ray Tracing, GPU, Interactive Rendering

The radiosity algorithm [3] is a technique for the estimation of exitant radiance in an environment, a view-independent model based on the finite-element method that yields illumination leaving one surface and reaching another. The technique is principally used in the context of diffuse reflectors. These surfaces are approximated via a finite number of planar patches with constant radiosity and reflectivity; a form-factor denotes the fraction of energy leaving one patch and arriving at another. In order to compute the form-factor for a patch, the visibility between the patch and all patches over the hemisphere of directions above the patch must be determined. This is typically carried out via projection techniques [4], computationally expensive operations which constrain surfaces to static configurations in real-time scenarios.

This series of projects aims to investigate modern formulations of the radiosity algorithm that take advantage of GPU programmable shaders, including hybrid methods that combine point-based sampling and Monte Carlo methods with the traditional radiosity algorithm [5]. In particular, these projects will investigate (i) an efficient scene representation that is amenable to the radiosity algorithm and can be efficiently and dynamically (re)generated using GPU programmable shaders or GPGPU methods; (ii) GPU/SMP-based ray-casting and spatial-hashing methods [1, 2, 6] to estimate the form-factors in the radiosity solution (iii) point-based methods for first-bounce (primary ray intersection) light estimation on generated patches; (iv) various degrees of physical correctness for the respective models in view of their potential applications (e.g. architecture, video-games, cultural heritage, etc).

Bibliography


Efficient rendering of shadowcasting light sources

Supervised by: Keith Bugeja, Sandro Spina

Keywords: Computer Graphics, Ray Tracing, GPU, Interactive Rendering

In the field of interactive rendering, a number of ad-hoc illumination models are often employed to balance the performance-quality trade-off. Visibility testing helps determine whether a surface is in shadow or not with respect to a particular light source and is often a very time-consuming part of an interactive rendering implementation. Thus, a very common trade-off is the use of light sources that do not cast shadows, boosting performance at the cost of image quality. The classification of light sources into shadowcasting and non is often-times arbitrary and left to the designers of a particular virtual environment.

This series of projects aims to investigate novel methods for the automatic per-frame classification and rendering of light sources based on two criteria: budget and image quality. In particular, for a given rendering budget, the method(s) should classify as shadowcasters the light sources that affect the image most, with respect to perceived differences from a reference image [3].

The use of point light sources to simulate radiosity in indirect lighting solutions provides a unified mathematical framework for the problem of global illumination and reduces the full light transport simulation to the calculation of direct illumination from many virtual light sources [1]. The many-lights formulation is scalable: an image can be generated in a fraction of a second or allowed to converge to a full solution over time. Virtual point lights are generated by tracing light source emissions, recording photon-medium intersections and the respective energy transfers [2]. Following the classification stage, the use of many-lights algorithms in visualising shadowcasting light sources will be investigated.

Bibliography


Simulating formally specified language behaviour and analysis

Supervised by: Adrian Francalanza
Keywords: Semantics, Process Calculi, Programming Language Design, Concurrency

It is now standard for language designers to formally specify the runtime semantics of a programming language using operational semantics and the static semantics using a type system [2]. It is however a laborious task to get this semantics right and establish properties about it e.g., subject reduction or progress.

PLT Redex [1] is a domain-specific language designed for specifying and debugging operational semantics. From standard reduction rules and (type) derivation rules, PLT Redex allows the designer to interactively explore terms and to use randomized test generation to attempt to falsify properties of the semantics. This allows the language designer to tighten the feedback loop and vet out obvious mistakes cheaply through automated tests.

The candidate will be asked to understand an existing language whose (static and dynamic) semantics is already formally defined. She will then be asked to implement this specification in PLT Redex and analyse the behaviour of the language using the model implemented.

Bibliography


Algorithm Implementation on Highly Parallel Architectures

Supervised by: Johann A. Briffa  
Keywords: GPU, GPGPU, Parallel Computing, High Performance Computing

CUDA is an interface for general-purpose programming on Graphical Processing Units (GPUs) from NVIDIA. The architecture of GPUs emphasises massive parallelism of arithmetic units at the expense of control units and memory caching. This allows a very high speed-up for classes of computationally-intensive data-parallel problems, often found in scientific computing. This is an implementation project, with a significant research orientation.

Objectives: Implement a compute-intensive algorithm on GPUs and optimize for speed. Suitable algorithms to be discussed with supervisor; possible examples include algorithms used in high energy physics from an ongoing collaboration with the ALICE experiment at CERN. Examples of prior work by the supervisor can be seen in [2, 1].

Student background/interests:
- Interest in parallel computing.
- Aptitude and willingness to program (the project requires the use of C++ and NVIDIA CUDA; prior OO development experience in another language is suitable).

Bibliography


Updates to Distributed Simulator for Communications Systems

Supervised by: Johann A. Briffa  
Keywords: Software Engineering, High Performance Computing

SimCommSys is a multi-platform distributed Monte Carlo simulator for communication systems [1]. The error control coding component implements various kinds of binary and non-binary codes, including turbo, LDPC, repeat-accumulate, and Reed-Solomon. This code base has been in continuous development since 1997, and currently weighs in at over 55,000 physical lines of code, written by Dr Briffa and collaborators. The distributed computing component of this code uses a client/server architecture built on TCP/IP to facilitate running simulations on grid resources; this also works well on local clusters.

Objectives: This project will look to extend the existing code base, continuing our previous work in this area. Various extensions could be looked at, including:

- Writing a cross-platform GUI for the simulator (i.e. writing software to create and edit simulation files in a user-friendly way).
- Writing a back-end / middle-ware for matching resources with simulations.
- Adding a result validation component to confirm simulation reproducibility and facilitate the use of public computing.

Student background/interests:

- Interest in low-level computing issues and parallel computing.
- Aptitude and willingness to program (the project requires the use of C++; prior OO development experience in another language is suitable).

Bibliography

Machine learning methods for handling missing data in medical questionnaires

Supervised by: Lalit Garg

Keywords: Business Intelligence, Artificial Intelligence, Machine Learning, Missing data analysis, Tensor decomposition

Self-report questionnaires[4] are used as an extremely valuable instrument to assess the quality of life of a patient, its relationship with socioeconomic and environmental factors, disease risk/progress, treatment and disease burden, treatment response and quality of care. However, a common problem with such questionnaires is missing data. Despite enormous care and effort to prevent it, some level of missing data is common and unavoidable. Such missing data can have a detrimental impact on statistical analyses based on the questionnaires responses. A variety of methods [4] have been suggested for missing data imputation. Nevertheless, more research is desperately needed to assess and improve the reliability of data imputation. In this project [4, 1, 3, 2], we will develop a novel algorithm/method which would evaluate all existing algorithms (including those we developed recently [4]) to decide the best algorithm based on the correlation and dependence among (observed) values and other parameters. This way we would be able to ensure the best prediction/imputation of missing values based on known values. This way an application will be developed to effectively impute missing data initially implemented using Matlab which can then be developed as an independent package or a web-based tool.

Bibliography


Real-time hospital bed occupancy and requirements forecasting

Supervised by: Lalit Garg

Keywords: Business Intelligence, Artificial Intelligence, Machine Learning, Healthcare Modelling, Resource Optimization, Forecasting, Data Mining, Modelling And Simulation

Healthcare resource planners need to develop policies that ensure optimal allocation of scarce healthcare resources [3, 2, 1]. This goal can be achieved by forecasting daily resource requirements for a given admission policy. If resources are limited, admission should be scheduled according to the resource availability. Such resource availability or demand can change with time. We here model patient flow through the care system as a discrete time Markov chain [3, 2, 1]. In order to have a more realistic representation, a non-homogeneous model is developed which incorporates time-dependent covariates, namely a patient’s present age and the present calendar year [3, 2, 1]. However, these models use historical data to estimate parameter values to be used in the model. These parameter values may not always represent the present scenario. As the healthcare system is a very dynamic and complex system it is necessary to continuously assess and update the parameter values to correctly represent the present scenario. We would enhance models developed earlier [3, 2, 1] to facilitate real-time assessment and corrections of parameter values to well reflect and accommodate changes in the scenario. We would develop an algorithm which will continuously estimate the parameter values, and monitor the change in the parameter values and develop the new model with the updated parameters to update resource requirements forecast. A real dataset of patients admitted in the Mater Dei Hospital would be used to evaluate the proposed methods/procedures/algorithms developed.

Bibliography


Supervisor Index

Azzopardi, George, 40
Briffa, Johann A., 39, 53, 54
Bugeja, Keith, 6, 8, 20, 28, 48, 50
Colombo, Christian, 11, 13, 15
Debono, Carl James, 29–32
Ebejer, Jean-Paul, 9, 36, 38
Ellul, Joshua, 6, 20, 22, 24, 26, 28, 46
Farrugia, Reuben, 33, 35
Francalanza, Adrian, 17, 19, 52
Garg, Lalit, 55, 57
Pace, Gordon, 9, 11, 13, 15
Spina, Sandro, 8, 48, 50
Vella, Kevin, 6, 20, 28, 44, 48
Vella, Mark, 44, 46
Keyword Index

Artificial Intelligence, 30, 32, 55, 57
Big Data, 38
Bioinformatics, 36, 38
Business Intelligence, 55, 57

Cheminformatics, 36
Cloud Computing, 29
Compilers, 22, 24, 26
Computer Graphics, 6, 8, 29, 31, 48, 50
Computer Vision, 35, 40
Concurrency, 17, 52
Cyber Security, 44, 46

Data Compression, 31
Data Mining, 57
Data Networks, 32
Data Science, 36, 38
Digital Forensics, 44, 46
Digital Image Processing, 30, 33, 39, 40

Embedded Systems, 39
Forecasting, 57

Game Playing, 9
Genetic Algorithms, 6
GPGPU, 53
GPU, 8, 48, 50, 53
Graph Theory, 32

Healthcare Modelling, 57
High Performance Computing, 40, 53, 54

Image Restoration, 33
Interactive Rendering, 48, 50

Logic, 19

Machine Learning, 40, 55, 57
Missing data analysis, 55

Modelling And Simulation, 57
Multi-cores, 20

Operating Systems, 22, 24, 26, 28
Parallel Computing, 53
Pointclouds, 8
Process Calculi, 52
Programming Language Design, 52
Ray Tracing, 6, 8, 48, 50
Resource Optimization, 57
Runtime Analysis, 17
Runtime Verification, 11, 13, 15, 19
Semantics, 52
Software Engineering, 39, 54
Super-Resolution, 33
System Software, 20
Systems, 22, 24, 26, 28
Tensor decomposition, 55
Video Processing, 29–31