6th INTERNATIONAL NORTH-AMERICAN CONFERENCE ON INTELLIGENT GAMES AND SIMULATION

GAMEON-NA 2011

3rd INTERNATIONAL NORTH AMERICAN SIMULATION TECHNOLOGY CONFERENCE

NASTEC 2011

EDITED BY

Mei Si

SEPTEMBER 28-30, 2011

RENSSELAER POLYTECHNIC INSTITUTE
TROY, USA

A Publication of EUROSIS-ETI

Printed in Ghent, Belgium

6th International North-American Conference on Intelligent Games and Simulation

3rd North American Simulation Technology Conference

TROY, USA

SEPTEMBER 28-30, 2011

Organized by

ETI

Sponsored by

EUROSIS

Rensselaer Polytechnic Institute

Co-Sponsored by

Ghent University	**Larian Studios**
UBISOFT	**GR@M**
GAME-PIPE	**The MOVES Institute**
Model Benders LLC	**Binary Illusions**
University of Skovde	**BITE**

Higher Technological Institute

Hosted by

Rensselaer Polytechnic Institute

Troy, USA

PROGRAMME COMMITTEE

Peripheral

Psychology and Affective Computing
Bill Swartout, USC, Marina del Rey, USA

Artistic input to game and character design
Richard Wages, Nomads Lab, Koln, Germany

Storytelling and Natural Language Processing
Ali Arya, Carleton University, Ottawa, Canada
Jenny Brusk, Gotland University College, Gotland, Sweden
R. Michael Young, Liquid Narrative Group, North Carolina State University, Raleigh, USA
Clark Verbrugge, McGill University, Montreal, Canada

Modelling of Virtual Words
Ruck Thawonmas, Ritsumeikan University, Kusatsu, Shiga, Japan

Online Gaming and Security Issues in Online Gaming
Pal Halvorsen, University of Oslo, Oslo, Norway
Andreas Petlund, University of Oslo, Oslo, Norway
Jouni Smed, University of Turku, Turku, Finland
Knut-Helge Vik, University of Oslo, Oslo, Norway

MMOG's
Chris Joslin, Carleton University, Ottawa, Canada
Michael J. Katchabaw, The University of Western Ontario, London, Canada
Alice Leung, BBN Technologies, Cambridge, USA
Mike Zyda, USC Viterbi School of Engineering, Marina del Rey, USA

Serious Gaming

Wargaming Aerospace Simulations, Board Games etc....
Roberto Beauclair, Institute for Pure and Applied Maths., Rio de Janiero, Brazil
Henry Lowood, Stanford University Libraries, Stanford, USA
Jaap van den Herik, Tilburg University, Tilburg, The Netherlands

Games for training
Michael J. Katchabaw, The University of Western Ontario, London, Canada
Gustavo Lyrio, IMPA, Rio de Janeiro, Brazil
Tony Manninen, University of Oulu, Oulu, Finland
Martina Wilson, The Open University, Milton Keynes, United Kingdom

Games Applications in Education, Government, Health, Corporate, First Responders and Science
Paul Pivec, RaDiCAL
Daniela M. Romano, University of Sheffield, Sheffield, United Kingdom
Russell Shilling, Office of Naval Research, Arlington VA, USA

Games Interfaces - Playing outside the Box

Games Console Design
Chris Joslin, Carleton University, Ottawa, Canada
Anthony Whitehead, Carleton University, Ottawa, Canada

PROGRAMME COMMITTEE

Mobile Gaming

Stefano Cacciaguera, University of Bologna, Bologna, Italy
Sebastian Matyas, Otto-Friedrich-Universitaet Bamberg, Bamberg, Germany

Perceptual User Interfaces for Games

Tony Brooks, Aalborg University Esbjerg, Esbjerg, Denmark
Lachlan M. MacKinnon, University of Abertay, Dundee, United Kingdom

Gaming Robots

Leon Rothkrantz, Delft University of Technology, Delft, The Netherlands

NASTEC

Conference Chair

Mokhtar Beldjehem, St.Anne's University, Nova Scotia, Canada

Honorary Chairs

Lotfi A. Zadeh, Berkeley University, CA, USA
Ronald Yager, Iona College, New Rochelle, USA
Madan M. Gupta, University of Saskatchewan, Saskatoon, Saskatchewan, Canada
Hojjat Adeli, The Ohio State University Columbus, OH, U.S.A.
I. Burhan Turksen, University of Toronto, Toronto, Canada

International Programme Committee

Ajith Abraham, Norwegian University of Science and Technology, Norway
Hojjat Adelli, Ohio State University,Columbus, OH, USA
Esma Aimeur, University of Montreal, Montreal, Canada
Troels Andreasen, Roskilde University,Roskilde, Denmark
Riad Assied, Petra University, Amman, Jordan
Bilal M. Ayyub, University of Maryland College Park, MD, USA
Mourad Badri, University of Quebec Trois-Rivieres, Canada
Linda Badri, University of Quebec Trois-Rivieres, Canada
Valentina Emilia Balas, Aurel Vlaicu University of Arad, Arad, Romania
Marek Balazinski, Ecole Polytechnique de Montreal, Montreal, Canada
Ildar Batyrshin, Kazan State Technological University, Kazan(Tatarstan), Russia
Nabil Belacel, National Research Council, New Brunswick, Canada
Mohamed Bettaz, INI/MESRS, Algiers, Algeria
Prabir Bhattacharya, Concordia University, Montreal, Canada
Loredana Biacino, Universita degli Studi di Salerno, Salerno, Italy
Ranjit Biswas, Institute of Technology & Management, Gurgaon, India
Ulrich Bodenhofer, Johannes Kepler University, Linz, Austria
Piero P. Bonissone, General Electric, USA
Gloria Bordogna, Istito par le Technologie Informatche Multimediali, Milano, Italy
Patrick Bosc, ENSSAT, University of Rennes, France
Boubaker Boufama, University of Windsor, Windsor, Canada
Mounir Boukadoum, UQAM, Montreal, Canada
Ivan Bruha, McMaster University,Hamilton, Ont., Canada
Pascal Bruniaux, ENSAIT, Roubaix, France
Bill P. Buckles, University of North Texas, USA
Liberato Camilleri, University of Malta, Msida, Malta
Joao P. Carvalho,INESC-ID, Lisboa University,Lisboa, Portugal
Allaoua Chaoui, University of Constantine, Algeria
Guanrong (Ron) Chen, City University of Hong Kong, China
Agnieska Cichocka, Unvesity of Lodz, Poland
Alain Colmerauer, University of Aix-Marseille II, Marseille, France
Michel Dagenais, Ecole Polytechnique de Montreal, Montreal, Canada

PROGRAMME COMMITTEE

PROGRAMME COMMITTEE

Witold Pedrycz, University of Edmonton, Edmonton, Alberta, Canada
James F. Peters III, University of Manitoba, Canada
Henri Prade, University of Paul Sabatier, Toulouse, France
Shahram Rahimi, Southern Illinois University,Carbondale, Il., USA
Djamal Rebaine, UQAC, Chicoutoumi, Canada
Marek Reformat, University of Edmonton, Edmonton, Alberta, Canada
Burghard B. Rieger, University of Trier, Trier, Germany
Stuart H. Rubin, Space and Naval Warfare Systems Center, USA
Daniel Rodríguez, University of Alcalá, Madrid, Spain
Imre J. Rudas, Budapest Technical University, Budapest, Hungary
Houari Sahraoui, University of Montreal, Montreal, Canada
Aziz Salah, UQAM, Montreal, Canada
Johann Schumann, RIACS/NASA Ames, USA
Antoaneta Serguieva, Brunel University, West London, United Kingdom
Siti Mariyam Shamsuddin, Universiti Teknologi Malaysia, Malaysia
Pierre Siegel, University of Aix-Marseille I, Marseille, France
Constantinos I. Siettos, National Technical University of Athens, Athens, Greece
Nematollaah Shiri, Concordia University, Montreal, Canada
James F. Smith, III, Naval Research Laboratory, Washington, DC, USA
Roman Slowinski, Poznan University of Technology, Poznan, Poland
Mu-Chun Su, National Central University, Taiwan, China
Dutta Sumitra, INSEAD, Fontainebleu, France
M.N.S. Swamy, Concordia University, Montreal, Canada
Hideyuki Takagi, Kyushu University, Fukuoka, Japan
Hamid R. Tizhoosh, University of Waterloo, Canada
Jose A. B. Tome, INESC-ID, Lisboa University, Lisboa, Portugal
Enric Trillas, European Centre for Soft Computing, Mieres(Asturias), Spain
Edward Tsang, University of Essex, United Kingdom
Hans Vangheluwe, McGill University, Montreal, Canada
Athanasios Vasilakos, University of Western, Macedonia,Greece
Xizhao Wang, HeBei University, China
Ronald R. Yager, IONA College, New Rochelle, N.Y., USA
Takeshi Yamakawa, Kyushu Institute of Technology, Kyushu, Japan
Mustapha Yassine, National University of Amman, Amman, Jordan
Ting Yu, University of Sydney, Sydney, Australia
Nevin Vunka Jungum, University of Mauritius, Mauritius
Sahnoun Zaidi, Universite de Constantine, Algeria

Reality Mining and Surprise Modelling

Danny Van Welden, KBC, Brussels, Belgium

Industrial Simulation

Guodong Shao, NIST, Gaithersburg, USA

Ecological sustainable development and Innovative technologies for a bio-based economy

Philippe Geril, ETI Bvba, Ostend, Belgium

GAME'ON-NA 2011

NASTEC'2011

All articles published in these Proceedings have been peer reviewed

EUROSIS-ETI Publications are ISI-Thomson and INSPEC referenced

A CIP Catalogue record for this book is available from the Royal Library of Belgium under nr.12620

For permission to publish a complete paper write EUROSIS, c/o Philippe Geril, ETI Executive Director, Greenbridge NV, Wetenschapspark 1, Plassendale 1, B-8400 Ostend, Belgium.

EUROSIS is a Division of ETI Bvba, The European Technology Institute, Torhoutsesteenweg 162, Box 4, B-8400 Ostend, Belgium

Printed in Belgium by Reproduct NV, Ghent, Belgium
Cover Design by Grafisch Bedrijf Lammaing, Ostend, Belgium
Cover Pictures by Larian Studios, Ghent, Belgium

EUROSIS-ETI Publication

ISBN: 978-9077381-65-6
EAN: 978-9077381-65-6

Preface

On behalf of the Rensselaer Polytechnic Institute I would like to welcome you to GAMEON-North America 2011-NASTEC’2011, and to the fair city of Troy.

This year’s event, smaller in size than normal no doubt because of the present economic climate, nevertheless succeeds in bringing together a number of presentations covering such wide-ranging fields as Game Methodology and AI, Game Graphics, Motion in Graphics, Serious Gaming and Industrial Simulation offering the participants a broad overview of present-day gaming and simulation research.

As well as the peer-reviewed papers, Game-On 'NA 2011-NASTEC’2011 features a number of invited talks highlighting research done in the field of computer gaming. These talks are by Heidi Boisvert, Creative Director & Founder, futurePerfect Lab and by David Thue of the University of Alberta, Canada.

To finish of the research theme we will visit the Games Labs at Rensselaer Polytechnic Institute.

Again, welcome to Troy and have a good conference.

Mei Si
Conference Chair
Rensselaer Polytechnic Institute
Troy, USA

CONTENTS

CONTENTS

SCIENTIFIC PROGRAMME

GAME METHODOLOGY AND AI

PROCEDURAL GENERATION OF SOKOBAN LEVELS

Joshua Taylor and Ian Parberry
Dept. of Computer Science & Engineering
University of North Texas
Denton, TX, USA
Email: ian@unt.edu, JoshuaTaylor@my.unt.edu

KEYWORDS
Procedural generation, Sokoban, puzzle.

ABSTRACT

We describe an algorithm for the procedural generation of levels for the popular Japanese puzzle game Sokoban. The algorithm takes a few parameters and builds a random instance of the puzzle that is guaranteed to be solvable. Although our algorithm and its implementation runs in exponential time, we present experimental evidence that it is sufficiently fast for offline use on a current generation PC when used to generate levels of size and complexity similar to those human-designed levels currently available online.

INTRODUCTION

In puzzle games the level design can make the difference between a game that is trivially easy or completely impossible. It is difficult to find the balance between the two, where the levels are challenging but still solvable. Here we present an algorithm that automates the generation of Sokoban puzzles of a given difficulty.

Sokoban is a puzzle game played on a rectangular grid. The goal is for the player's avatar to push boxes onto marked goal squares. The challenge comes from the placement of the walls, goals and boxes and the restriction that the avatar is only strong enough to push one block at a time and cannot pull blocks at all. The simplest way of explaining it is to show a picture, for example Figure 1, which shows a level with a single box and a single goal. This figure and the other screenshots in this paper are from JSoko (Damgaard et al. 2010).

Culberson has shown (Culberson 1998) that Sokoban is PSPACE-complete, meaning that it is in a sense at least as difficult as almost any one-player game. (Most games that are hard in this sense are for two or more players.) This, together with its simple rules, makes Sokoban a challenging candidate for procedural generation of puzzle instances. Completely random Sokoban levels are extremely likely to be unsolvable, or if they are solvable, then they are likely to be very easy. Even hand-made levels suffer from this problem unless the person making the level is an experienced Sokoban level designer.

Most other research done on Sokoban has been geared towards solving existing Sokoban puzzles (Junghanns and Schaeffer 1997, Botea et al. 2003). Some work has also been done on estimating the difficulty of a given Sokoban problem (Jarušek and Pelánek 2010a, Ashlock and Schonfeld 2010). Relatively little research has been done on generating new Sokoban levels (Murase et al. 1996, Masaru et al. 2003), although there are several existing generator programs (Mühendisi Accessed 2011). Additionally, there has been some research on generating levels for other PSPACE-complete puzzle games (Servais 2005).

The interested reader is invited to visit our Sokoban Generator webpage (Taylor and Parberry 2011) for supplementary information. This includes some more detailed instructions for the novice on how to play Sokoban, several hundred procedurally generated Sokoban levels, a link to an open source Java implementation called JSoko on which to play-test those levels, a short video showing JSoko's solution to some of our levels, some larger color images from this paper, and the archived data from the experiments performed to generate the performance data for the tables and figures that will appear later in this paper.

OBJECTIVES

Any procedural generation system should satisfy several criteria (Doran and Parberry 2010): *novelty*, *structure*, *interest*, *controllability* and *speed*. Our Sokoban level generator possesses these qualities as follows: Novelty: The generator produces a new and different puzzle on each run. Structure: The puzzles are nontrivial yet not impossible to solve, without requiring verification of this by use of an automated solver. Interest: Players should find the prospect of solving the puzzles attractive. This is left for future work; Development is currently underway. Controllability: Designers have control over the size and difficulty of the generated levels. Speed: The generator can run offline on a modest computer and generate at least one challenging puzzle, or hundreds of nontrivial puzzles per day.

Our primary aim is to generate reasonably difficult, but not impossible, Sokoban levels. There are two reasons for this. Firstly, these levels are the kind that are hard for a human to make, at least without a lot of experience. Secondly, we believe that in puzzle games, difficulty is related to interest.

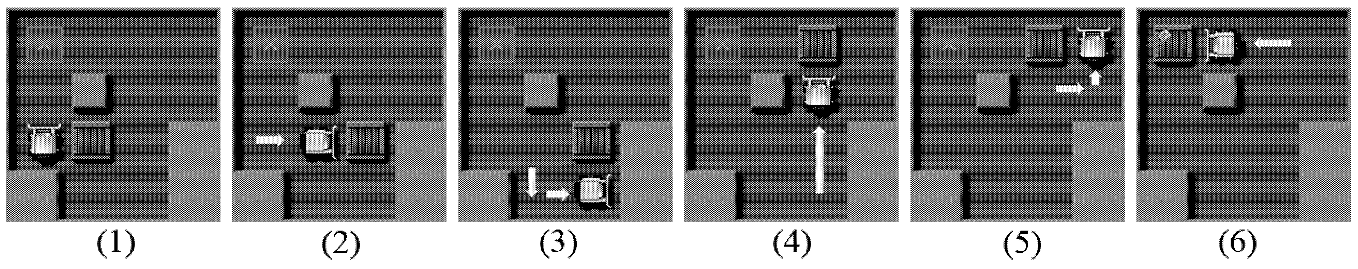


Figure 1: Solving a simple Sokoban level. The aim is to push the box to the square marked with the “X” at top left using the yellow bulldozer. The white arrows indicate player actions. The six images show, from left to right, (1) the start configuration, (2) push the box one place right, (3) reposition the player below the box, (4) push the box two places up, (5) reposition the player to the right of the box, and (6) push the box two places to the left into the final configuration.

Interesting puzzles are neither too difficult nor too easy (at least for most players), and yet it is these puzzles that are the most difficult to generate.

METHOD

The idea of working backwards from the goal towards the start is not new (Takes 2007), but previously it has only been used to *solve* existing levels. Here we use that idea to *generate* new levels. Our algorithm consists of three high-level steps, each of which will be described in more detail in its own subsection below.

1. Build an empty room.
2. Place goals in the room.
3. Find the state farthest from the goal state.

EMPTY ROOMS

To build an empty room, we use a method somewhat similar to that of (Murase et al. 1996). We begin by choosing a width and height for the level. This is done by simply picking a random number within a user-specified range. The level is then partitioned into a grid of 3×3 blocks. Each block is then filled in using a randomly chosen and randomly rotated or flipped template. The templates consist of a 3×3 pattern of walls and floors surrounded by a border of blanks, walls and floors (see Figure 2). The borders cause neighboring templates to overlap. A non-blank tile must match any pattern it overlaps, whether it is placed before or afterwards.

This overlap helps to create interesting levels by preventing some bad configurations from being generated. For example, the pattern consisting of a single wall in the middle surrounded by a ring of floor will become a large dead-end unless there are at least two floor tiles adjacent to each other and that pattern. Since the templates are randomly rotated and flipped before being placed, this is very easy to enforce by simply placing two adjacent floor tiles in the border of that template and leaving the rest blank.

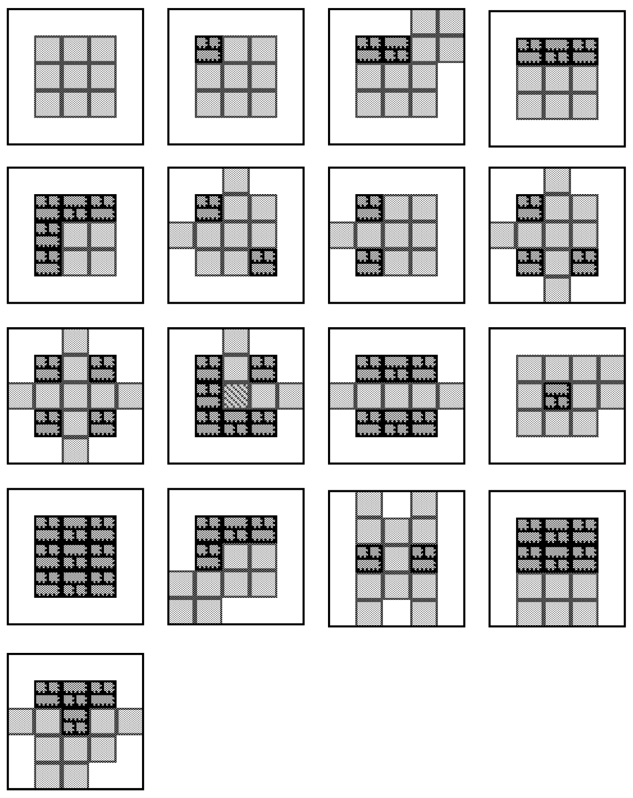


Figure 2: Templates used to design an empty room.

If the generator places blocks in such a way that it cannot fill in one of the cells with any of the available templates, it will discard that attempt and start over. The run time of this step is very small compared to the rest of the algorithm, so even throwing away several partial room shapes does not create any noticeable loss of speed.

Finally, there are some post-processing checks to make sure the level will work well with the remaining steps. Any level that fails one or more of these tests is discarded.

- The level is checked for connectivity. There should be one contiguous section of floor. There is one special case here. The templates that allow the player to pass through, but will not allow a box to pass, are checked as if there was a wall tile separating the two sides. This only affects this check, and that tile is counted as a floor tile in all other cases.

- Any level that has a 4×3 or 3×4 (or larger) section of open floor is discarded. By observation, such levels tend to make levels with very bushy, but not very deep state spaces. This makes it very hard to generate the level, but not much harder to solve it.
- The level must have enough floor space for the planned number of boxes, plus the player and at least one empty space.
- If the level contains any floor tiles surrounded on three sides by walls, it is discarded. This is a somewhat aesthetic choice, but such tiles are either obviously dead space if there is no goal there, or an easy place to get boxes out of the way otherwise, so we think it improves the quality of the resulting levels somewhat.

PlACING GOALS

Goal placing is done by brute force, trying every possible combination of goal positions. This is admittedly very inefficient. Many human made levels place the goals in certain patterns, such as a rectangle of contiguous goals, but by doing a brute force search for the best places to put the goals, some obvious patterns emerge.

One pattern that seems to hold for most, but not quite all, of the levels generated so far is that the goals are touching either a wall or another goal. Whether forcing this would provide a significant speed-up, or a significant drop in the quality of the resulting levels has not yet been investigated.

Our generator uses a timer that checks it has exceeded its allotted time. If it has, it will terminate and return the best result so far. To help ensure that that result is something interesting, even if not the best, the positions for the goal crates are checked in random order. This is done by creating a shuffled list of the empty spaces on the board.

FARTHEST STATE

For each placement of the goals the system finds the farthest state from that goal state, that is, the state with the longest shortest path from itself back to the goal. Over all goal states, the farthest farthest state is returned as the output of the generator. Thus, the distance from the goal state to the start state is the metric by which we judge the resulting levels, as well as influencing the algorithm used to search the state space. The definition of *distance* is crucial. In Sokoban there are four common distance metrics. The simplest is just the move count, incremented every time the avatar moves. As a measure of the difference between states, this does not work very well. Just making a large labyrinth with only one obvious solution will still give a high distance, but will be fairly trivial in the end.

The number of box pushes, incremented each time the avatar moves into a square containing a box, is not much different than the move count. A level that required the player to push boxes down long hallways would give a high score, but again would not be difficult, just tedious.

The box lines metric is more interesting. It counts how many times the player pushes a box, but any number of pushes of the same box in the same direction only count as a single box line. From our observations, the number of box lines corresponds fairly well with the difficulty of the resulting level. We are currently using the box line metric in our generator.

The last metric is box changes. It counts how many times the player stopped pushing one box, in any direction, and began pushing another. This may be an even better measure of difficulty, and may improve the overall speed of the generator, but it is more difficult to implement.

Any metric except for the move count allows us to abstract out the avatar position. Instead of keeping up with which square the avatar is in, we keep up with which group of contiguous floor squares it could reach. This abstraction provides a significant decrease in the time it takes to generate the set of further states.

All of this is done in reverse compared to how Sokoban is played. The reason for this is to prevent the generator from having to consider invalid moves. Any state reachable when moving in reverse will be solvable when played normally.

Unfortunately, none of the usual search algorithms are suitable for this problem. The most obvious way to find the farthest state is to use a breadth-first search, returning the last state found, but since moves in Sokoban are not reversible, the only way to prevent repetitions is to store a list of all visited nodes. For Sokoban, or any other PSPACE-complete problem, this will quickly fill up the available memory. Iterative deepening is unsuitable for similar reasons. Informed searches, like A* or IDA*, are unsuitable because the target is very vaguely defined, meaning we have no clear indication when to stop the search. To get around these problems, we use a form of iterative deepening twice, trading off the high memory requirements for a somewhat slower algorithm.

```
proc Go(goal) $\equiv$
  startSet := MakeStartSet(goal);
  resultSet := startSet;
  depth := $1$;
  do
     prevSet := resultSet;
     resultSet := Try(startSet, resultSet, depth);
     if resultSet $= \emptyset$ then exit fi;
     depth := depth $+ 1$;
  od;
  Go := (prevSet, depth).
```

```
proc Try(startSet, prevResults, depth) ≡
  resultSet := Expand(prevResults);
  tempSet := startSet;
  for $i := 1$ to depth do
    resultSet := resultSet − tempSet;
    tempSet := Expand(tempSet);
  od;
  Try := resultSet.
```

MakeStartSet takes the goal state and places the player into each available contiguous floor area. Expand takes a set of states and returns the set of states one step farther. What those states are depends on which metric is being used, which is why the choice of metric has such an impact on the running time. Go is almost a standard iterative deepening algorithm. It takes the goal, calls MakeStartSet to set things up and then calls Try one depth at a time. What is different here is the end condition. Go calls Try until it fails and then returns the previous set of results. Try takes the starting set, the previous results and a target depth. It then calls Expand on the previous results. Then it starts over expanding the start set, subtracting that from the results. What is left after the target depth has been reached is all of the nodes that can be reached in depth steps, but no sooner.

GENERATING LEVEL SETS

Our generator returns the set of all levels that are as far from the goal state as possible. This can be anywhere from one to a few hundred levels, and some are obviously better than others. We have an additional layer over the generator that attempts to select a good level from those generated and then collects the results of several runs into a level set. Additionally, it makes an attempt to reject levels that are much too easy, or are too similar to levels already in the level set. Finally, when the target number of levels has been reached, it attempts to sort them by some measure of difficulty.

The questions of what is *better* and what is *not good enough* both rely on the same, rather arbitrary, measure. We take the candidate level and give it a score based on a number of factors. To begin with, the score is $100 \cdot (\text{pushes} - \text{number of sibling levels} + 4 \cdot \text{lines} - 12 \cdot \text{boxes}) + \text{Random}(0, 300)$, where pushes is the number of box pushes in the solution, sibling levels is the number of other levels the generator found at the same depth, lines is the number of box lines in the solution, boxes is the number of boxes in the level and Random is just a random number between the given values. While this is, again, fairly arbitrary, the rationale is that both more pushes and more lines make the level more difficult, while levels with many sibling levels seem to be less interesting, just by observation. The number of boxes is subtracted not because more boxes makes the level less interesting, but because the number of lines needs to exceed the number of boxes by a certain factor for the level to have a better chance of being a good level. The random factor is mainly there to break ties.

Some other checks are made after getting the base score. Any trapped box is worth -100000 points, which is almost guaranteed to get the level rejected. Any box touching a wall is worth -150 points, a box touching the player is worth 50 points, and a box touching another box is worth 30 points. Finally, a goal area touching a goal area is worth 30 points. These constants can be adjusted by the individual designer to suit his or her intuition about features possessed by good Sokoban levels. Any level with a final score of 0 or less is rejected. The base score is quite a bit higher than the scores for most of the various other checks though, so not many levels are rejected at this point. This same score is then used to choose the best level from those generated, assuming any are left.

Once a level has passed all of the other tests, the program checks to see whether or not it is too similar to another level already in the set. Currently, this just removes the player and checks for exact matches for all of its rotations and reflections. This still generates a few levels that a human would consider too similar, so there is still more work to be done here.

Assuming no other level is too similar, the level is added to the set. When the set gets to target size, it is sorted by difficulty and written to a file. The measure of difficulty we currently use is $\text{lines} \cdot \log \text{lines} + \log \text{time} - \text{lines}/\text{pushes}$. This is based on our observations that the number of box lines is the most important factor. time is the time taken to generate that level, in seconds, and is mainly used as a tie breaker. The lines/pushes factor is small correction that favors levels with shorter box lines rather than long corridors.

EXPERIMENTAL RESULTS

We have implemented and tested our new algorithm for the automatic generation of Sokoban levels. Figure 6 contains some screenshots of sample levels generated. The reader is invited to visit our Sokoban Generator webpage (Taylor and Parberry 2011), where he or she can download some of our level sets and try them out.

Our algorithm is certainly suitable for offline use in a level-a-day style game. In practice it can generate several levels over the course of a day depending on how much CPU power it is given and how large the desired levels are. The theoretical run time for the generation of one level is roughly

$$b \binom{s}{b}^2,$$

where b is the number of boxes and s is the number of empty spaces. This is feasible for a fairly small number of boxes that might occur in practice. We have been able to generate levels with 4 boxes within a 2×3 level outline within a few hours.

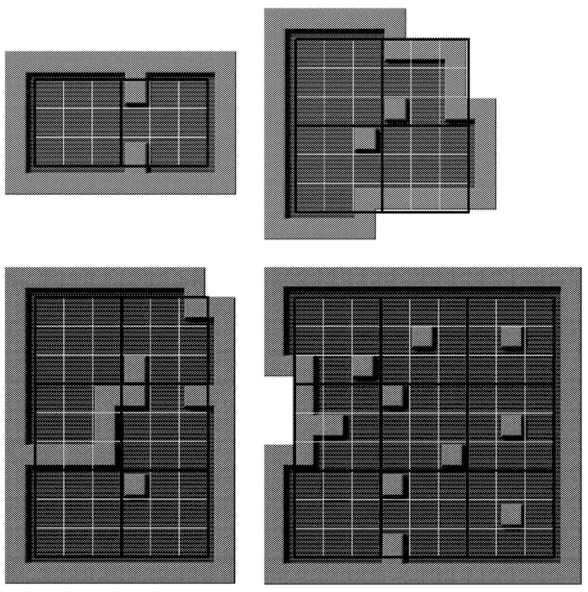


Figure 3: Examples showing the relative sizes of 1×2, 2×2, 2×3, and 3×3 Sokoban levels.

We ran experiments measuring the average run-time generating 10 random puzzles of each size 1×2, 2×2, 2×3, and 3×3. See Figure 3 for an indication of the relative sizes of these levels. All of these results are from runs on an Intel i7 3.2 GHz quad-core processor with hyperthreading. Our system is not written to make use of the extra cores, but we run several independent copies of the code simultaneously, relying on the operating system to place each copy on a separate processor core.

Tables 1 and 2 show in Column 3 the experimental running time required to generate 2-box and 3-box puzzles respectively, averaged over 10 samples for each entry. These data are depicted pictorially in Figure 4 (top) with an exponential trendline. Tables 1 and 2 also show in Column 2 the average number of moves required to solve the generated puzzles using the autosolver in JSoko, set to "move optimal with best pushes". These data are depicted pictorially in Figure 5 (bottom).

Table 3 shows in Column 3 the average experimental running time for 2×2 puzzles (which have 36 cells). See the level at top right of Figure 3 to get some idea of the size of the puzzle. These data are depicted pictorially in Figure 5 (top) with an exponential trendline. Table 3 also shows in Column 2 the average number of moves required to solve the generated puzzles using the autosolver in JSoko, set to "move optimal with best pushes". These data are depicted pictorially in Figure 4 (bottom).

Levels with a single box are generally uninteresting. Levels with 2 boxes can be generated very quickly, usually within a few seconds, but tend to be very easy. At 3 boxes the levels start to get slightly more interesting, and can still be generated within a few minutes. Levels with 4 or more boxes can

Size	Moves	Time
1×2	26	$<$ 1 sec
2×2	48	1.9 sec
2×3	60	16 sec
3×3	73	128 sec

Table 1: Average runtime for the generation of 2-box puzzles with the corresponding average number of moves needed to solve them (averaged over 10 random samples each).

Size	Moves	Time
1×2	38	58 sec
2×2	69	2.7 min
2×3	98	1.1 hr
3×3	115	24.5 hr

Table 2: Average runtime for the generation of 3-box puzzles with the corresponding average number of moves needed to solve them (averaged over 10 random samples each).

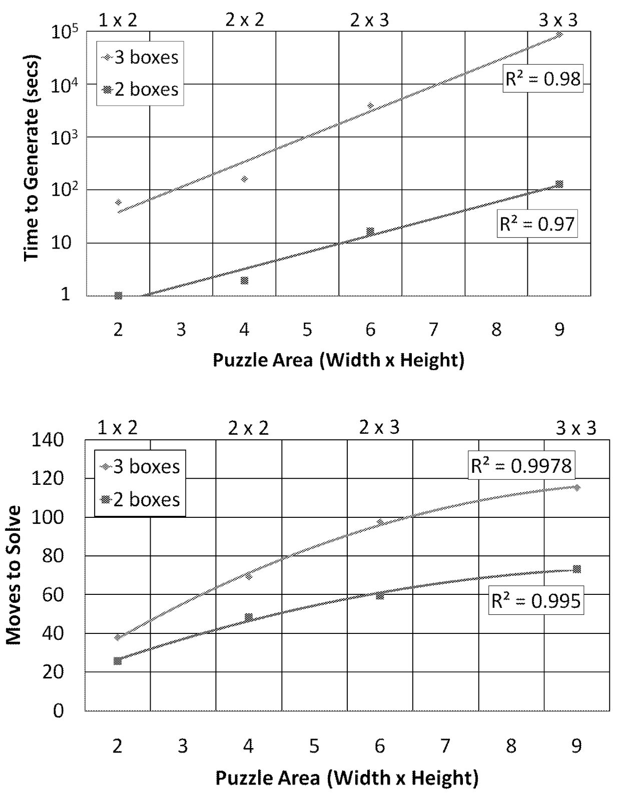
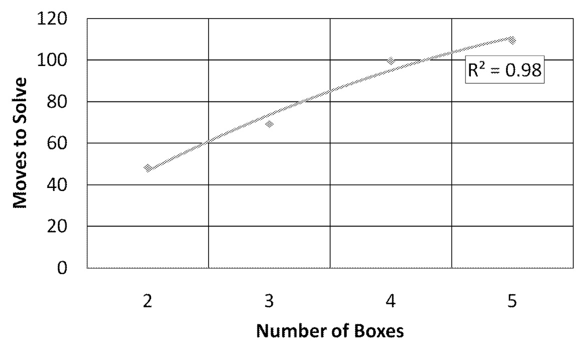


Figure 4: The average generation time in seconds for puzzles with 2 and 3 boxes versus puzzle area (top) and the number of moves needed to solve them (bottom).

Boxes	Moves	Time
2	48	1.9 sec
3	69	2.7 min
4	100	3.4 hr
5	109	26 hr

Table 3: The average experimental running time for the generation of 2×2 puzzles, with the corresponding average number of moves needed to solve them. The averages were computed over 10 random samples each.



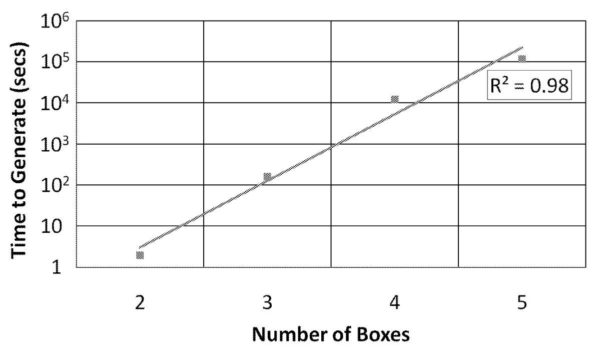


Figure 5: The average number of moves (top) and the average generation time in seconds (bottom) versus number of boxes for the generation of 2×2 puzzles.

be very interesting, and difficult, but take substantially more time to generate. Using the timer feature, we can force the generator to return the best result after a given time period. Using a time limit of 4 hours, we have generated levels with 5 and 6 boxes that appear interesting and difficult (for example Figure 6 includes some 5-box puzzles).

Using just iterative deepening, the algorithm runs several times faster, but uses much more memory. On some levels, the program crashed after consuming over 1.5GB of memory. Using our algorithm, the same levels never exceeded 40MB of memory.

CONCLUSION AND FUTURE WORK

While we found the puzzles that we generated "interesting", we provide no justification for this claim in this paper, although we do invite the reader to try for themselves by visiting our Sokoban Generator webpage (Taylor and Parberry 2011). We plan to gather data from play-testing in the next phase of this research, and we will report the results in a later paper. Some research into what makes a level interesting and what makes it difficult is needed, though some research on these questions has already been done (Ashlock and Schonfeld 2010, Jarušek and Pelánek 2010b).

REFERENCES

Ashlock D. and Schonfeld J., 2010. *Evolution for automatic assessment of the difficulty of Sokoban boards*. In *Proc. IEEE Congress on Evolutionary Computation*. 1–8.

Botea A.; Müller M.; and Schaeffer J., 2003. *Using Abstraction for Planning in Sokoban*. In *Computers and Games, Springer Lecture Notes in Computer Science*, vol. 2883. 360–375.

Culberson J., 1998. *Sokoban is PSPACE-complete*. In *Proc. International Conference on Fun with Algorithms*. 65–76.

Damgaard B.; Marxen H.; and Meger M., 2010. *JSoko*. URL `http://sourceforge.net/projects/jsokoapplet/`.

Doran J. and Parberry I., 2010. *Controlled Procedural Terrain Generation Using Software Agents*. *IEEE Transactions on Computational Intelligence and AI in Games*, 2, no. 2, 111–119.

Jarušek P. and Pelánek R., 2010a. *Difficulty Rating of Sokoban Puzzle*. In *Proc. Fifth Starting AI Researchers' Symposium*.

Jarušek P. and Pelánek R., 2010b. *Human Problem Solving: Sokoban Case Study*. Tech. Rep. FIMU–RS–2010–01, Faculty of Informatics, Masaryk Univ. Brno.

Junghanns A. and Schaeffer J., 1997. *Sokoban: A Challenging Single-Agent Search Problem*. In *Proc. IJCAI Workshop on Using Games as an Experimental Testbed for AI Research*. 27–36.

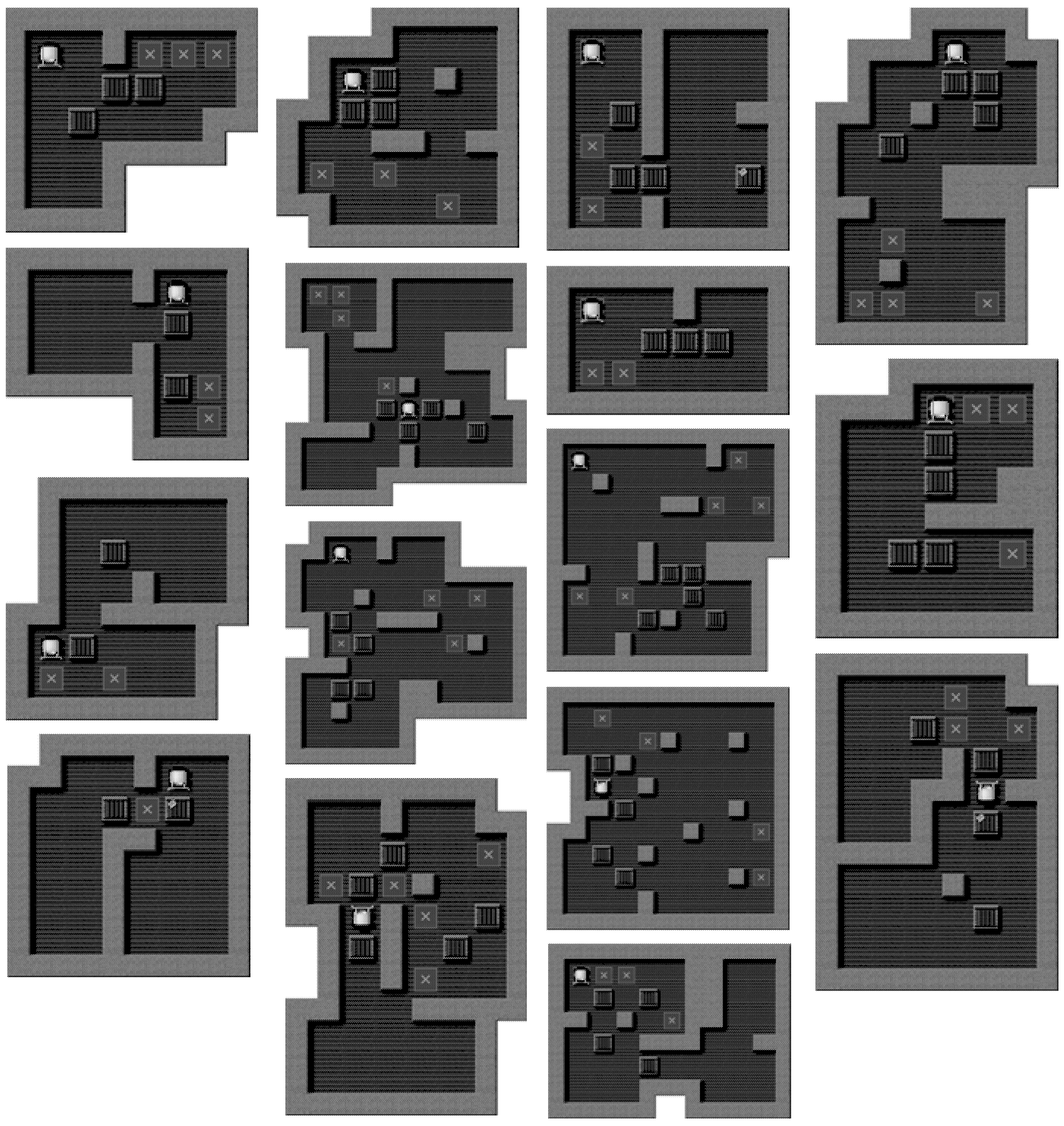


Figure 6: Some levels of varying difficulty created by our generator.

Masaru O.; Tomoyumi K.; and Satoru K., 2003. *A Method of Automatic Creation of Goal-Area in Sokoban Maps*. *Joho Shori Gakkai Shinpojiumu Ronbunshu*, 67–74.

Mühendisi M., Accessed 2011. *Sokoban Level Generators (A to Z)*. URL `http://www.erimsever.com/sokoban7.htm`.

Murase Y.; Matsubara H.; and Hiraga Y., 1996. *Automatic making of Sokoban problems*. In *PRICAI'96: Topics in Artificial Intelligence*, *Springer Lecture Notes in Computer Science*, vol. 1114. 592–600.

Servais F., 2005. *Finding Hard Initial Configurations of Rush Hour with Binary Decision Diagrams*. M.Sc. Thesis, Univ. libre de Bruxelles, Faculté des Sciences.

Takes F., 2007. *Sokoban: Reversed Solving*. Bachelor's Thesis, Leiden University.

Taylor J. and Parberry I., 2011. *Sokoban Generator*. URL `http://larc.unt.edu/ian/research/sokoban/`.

BIOGRAPHY

JOSHUA TAYLOR is a PhD student in the Department of Computer Science and Engineering at the University of North Texas. His research interests include procedural content generation.

IAN PARBERRY was born in London, England and emigrated as a child with his parents to Brisbane, Australia. After obtaining his undergraduate degree there from the University of Queensland he returned to England for a PhD from the University of Warwick. He has worked in academia in the US ever since. He is currently a full Professor in the Department of Computer Science and Engineering at the University of North Texas where he recently stepped down from a 2-year term as Interim Department Chair. A pioneer of the academic study of game development since 1993, his undergraduate game development program was ranked in the top 50 out of 500 in North America by The Princeton Review in 2010. He is on the Editorial Boards of the *Journal of Game Design and Development Education*, *IEEE Transactions on Computational Intelligence and AI in Games*, and *Entertainment Computing*, and he serves as the Secretary of the Society for the Advancement of the Science of Digital Games, which organizes the Annual Foundations of Digital Games conference. He is the author of 6 books and over 80 articles over 30 years' experience in academic research and education. His *h*-index is 18 and his Erdös number is 3. He can be contacted at `ian@unt.edu` or on Facebook. His home page is `http://larc.unt.edu/ian`.

DEPLOYING FUZZY LOGIC IN A BOXING GAME

Hamid Reza Nasrinpour, Siavash Malektaji, Mahdi Aliyari Shoorehdeli and Mohammad Teshnehlab
Department of Electrical and Computer Engineering
K. N. Toosi University of Technology
Seyedkhandan, Tehran, Iran
nasrinpour, siavashmalektaji@ee.kntu.ac.ir
m_aliyari, teshnehlab@eetd.kntu.ac.ir

KEYWORDS
Fuzzy Logic, Computer Game, Rule-Based System, Boxing

ABSTRACT

Nowadays computer games have become a billion dollar industry. One of the important factors in success of a game is its similarity to the real world. As a result, many AI approaches have been exploited to make game characters more believable and natural. One of these approaches which has received great attention is Fuzzy Logic. In this paper a Fuzzy Rule-Based System is employed in a fighting game to reach higher levels of realism. Furthermore, behavior of two fighter bots, one based on the proposed Fuzzy logic and the other one based on a scripted AI, have been compared. It is observed that the results of the proposed method have less behavioral repetition than the scripted AI, which boosts human players' enjoyment during the game.

INTRODUCTION

The academic definition of Artificial Intelligence (AI) states that AI is creating machines which can sensibly think and act as humans (Russell and Norvig 1995). This classic AI is usually concerned with the optimum solution to a problem and discusses how this solution can be found. Game AI is a code or technique which a computer uses to control Non-Player Characters (NPCs). It does not care how a computer makes a decision or thinks about the problem, it might use either a decision tree or a huge database, the point is just its reaction.

Game balance has been one of the considerable topics in Game AI. If a computer opponent always just followed similar patterns, or played too easily or strictly, the game would be annoying. Furthermore, the aim is not to defeat players, but rather to keep them amused. In fact, none of the opponents should be unbeatable.

In Game AI, natural game laws should be followed and cheating should be prevented as far as possible. Cheating AI is a term which refers to a situation where the AI has more information and advantages than the players. This is often implemented in games to increase the abilities of a machine and it might be acceptable if the player is not apprised of it. Some examples of cheating AI can be found in (Scott 2002). As a matter of fact, applying these cheats represents weaknesses and limitations of AI against human knowledge. Therefore it would be better to look for the methods which represent human knowledge, like Fuzzy rule-based systems, as they are a widespread form of Fuzzy logic (Zadeh 1965).

Fuzzy logic can easily rate any input based upon importance. Additionally it is really suitable to do multiple operations at once (Doss 2006). Human knowledge and the ways in which humans think and infer can be directly put into a game by Fuzzy rule-based systems. For instance, in a Fighting Game (Rollings and Ernest 2006), a game developer can take advantage of Fuzzy logic to reach higher levels of realism and human-level AI in behavior of the opponent's character. This makes the human players feel that they are playing against another human, not an omniscient computer, which knows about all of the angles and distances between itself and its opponent and consequently, makes no mistake unless it deliberately decides to make the game easier for the human opponent.

In recent years, we have witnessed many applications of Fuzzy techniques in the domain of games. The game *S.W.A.T. 2* has employed Fuzzy logic as an instance of action games (Johnson and Wiles 2001). In the genre of Real Time Strategy (RTS) games, *Civilization: Call to Power* uses Fuzzy State Machines (FuSM) as well (Johnson and Wiles 2001). The great performance of Fuzzy methods in 2009 simulated car racing championship can be found in (Loiacono et al. 2010). Another Fuzzy controller for a car racing championship is discussed in (Perez et al. 2009), and also a Fuzzy-based architecture has been tested in *The Open Car Racing Simulator (TORCS)* in (Onieva et al. 2010). There is a Fuzzy Q-learning method which has been implemented in the game of Pac-Man (DeLooze and Viner 2009). An agent-based Fuzzy system has been applied in the *Battle City* game into the bargain (Li et al. 2004).

In this paper, a method without cheating has been introduced for applying Fuzzy logic in a fighting game. In order to avoid cheating and make a natural and human-level AI, first the game engine was implemented independently. Then the game engine would give control of the characters to a human player or an AI in the same way. Moreover, the rules which the Fuzzy AI fires are the Fuzzy rules which can be followed easily by a human and do not have any complicated computational overhead.

BACKGROUND

Boxing is an old Atari 2600 video game released by Activision group in 1980. *Boxing* shows a top-down view of two boxers which can punch their opponent, as shown in Figure 1. The choice between punching hands (right or left) is made automatically and the human player just presses the hit button. Two boxers are always in front of each other and they can only move up, down, forward and backward. So they cannot rotate. The game finishes after two minutes or when one boxer knockouts the other one by giving him 100 punches.

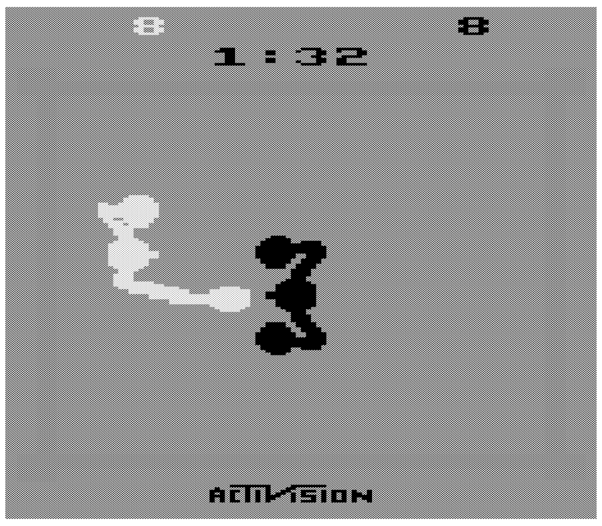


Figure 1: Original Atari Boxing Game

Clever Boxer, discussed in this paper, is an extension of the original *Boxing* game where boxers can rotate and go behind the opponent and the punching hand selection is made manually. So the human player must press the proper button. The boxers can make fast movements in normal directions (forward, backward, left and right) and if they try to punch while moving fast, their punch on the opponent will be more powerful. In addition, the boxer must have enough stamina to punch or to move fast.

GAME ENGINE

The game engine has been designed based on physical rules in order to be able to be implemented on boxer agent bots. Hence the game could be considered as an agent-based simulation software of a real boxer agent's behavior.

Design

This section covers some important rules of the game. The boxer agent, whose behavior is implemented by a Finite State Machine (FSM), has three different types of commandable states which can directly be given by a human or AI player, which are shown in Table 1. If a boxer is commanded to punch, it will stand still and throw a punch. But the commands of rotation and the commands of direction can be given and run simultaneously. The act of throwing a punch, implemented by a FSM, takes place in 4 states. In fact, in each state the hand of the boxer reaches out slightly in a way that in the 4th state the hand is completely stretched. When a punch hits the opponent, it will be effective, if and only if the hand is in the 3rd or 4th state. Besides, the powers of punches in 3rd and 4th states are different from each other.

There are some other states that are not directly commandable and a boxer could be in. An expected state of these non-commandable states is the *Overshooting* state in which the boxer is punched in its back or stomach. The human player or AI does not have any control over the boxer in *Overshooting* states. The complete list of non-commandable states with exact definitions, which implicitly explain the rules of the game, is shown in Table 2.

Table 1: Commandable States of Boxer Agent

State Type	State Name	
Direction States	1.Fix 2.GoingForward 3.GoingBackward 4.GoingRight 5.GoingLeft	6.GoingForwardFast 7.GoingBackwardFast 8.GoingRightFast 9.GoingLeftFast
Punching States	1.PunchRight 2.PunchLeft	3.NoPunching
Rotation States	1.RotateRight 2.RotateLeft	3.NoRotation

Table 2: Non-Commandable States of Boxer Agent

Name	Description
BeingPushed	The boxer is being hustled by the other one. It happens when a boxer is moving faster than the other one.
BoxerContact	Two boxers have a collision with each other.
CircularOvershooting	The boxer is punched in its eye, and then it rotates up to 45 degrees while overshooting.
DoubleOvershooting	If the back of a boxer come into contact with the ring border while overshooting, it will be in this state.
FastPunching	The boxer punches while fast moving. This punch is more powerful than a normal punch.
Overshooting	The boxer is punched in its back or stomach and the human player or AI cannot control it.
Punching	The boxer punches while normal moving.
RingAccelaration	The boxer pushes its back toward the ring border to go forward faster. If a boxer punches in this state, it will go to the *FastPunching* state.
RingContact	The body of the boxer has touched the ring border

As it was mentioned earlier, a boxer will be unable to punch or to move fast if its stamina is lower than a specific level. Also it will not be able to punch if it receives too many punches in its hand. In other words, every hand can endure up to a specific number of hits; otherwise the hand will lose its ability.

Implementation

The game framework, written in Microsoft Visual C#, provides a control loop driven by an external timer to handle animations and collisions. The framework also gives AI an opportunity to perform its own processing, while human players can asynchronously command their boxers.

If AI were reliant on the speed and precision of its punches rather than its playing strategy, it could be seen as cheating because it is impossible for a human player to replicate the computer's effort as fast or as easily as the computer. Therefore the game engine applies two different types of delay to AI in order to make it use more realistic game strategies and act like a human; one is the information delay, which is the delay of receiving data of the environment and the opponent's character from the game engine. The other one is the delay in execution after which a command from the AI is issued. The game engine informs AI of the nature of these delays.

AI

Two different AIs are implemented to control the boxer during the game. One is based on scripted AI and the other one is based on Fuzzy logic. In this section, these two implementations are discussed.

Scripted AI

Scripting (Bourg and Seeman 2004) is currently the most common means of control in Game AI. Most developers resort to scripts to implement Game AI for complex games where the number of choices at each turn varies from hundreds to even thousands. Some advantages of scripting are being understandable, easy to implement and easily extendable (Tozour 2002).

In this method, some scripts have been written for different situations which might happen during the game. For instance, when a boxer is too close to the opponent in a way that its punches are not effective, it pushes forward the opponent in a fast movement, or when its hand does not have enough stamina for punching, it keeps itself at a safe distance from the opponent, where the opponent's punches would not be efficient and looks forward to the timeout.

Furthermore, some parameters have been defined based on the AI character's and the opponent's health, stamina and abilities of their hands. A selection of these parameters has been used in some scripts. A simple example of the parameters' effects on the scripts is that the scripted AI will play its conservative scripts if the AI boxer's health is less than half of its opponent's health.

Fuzzy Rule-Based AI

Figure 2 provides an overview of the architecture of the game engine and its relation to the Fuzzy rule-based AI. It is based on a classical three-layer hierarchy: perception, control and actuation layer.

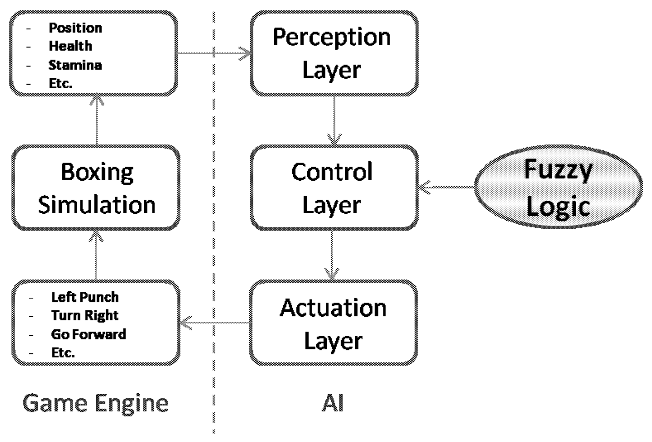


Figure 2: Overview of the system architecture

The perception layer, which receives the position, health, stamina and hand abilities of two boxers from the game engine, first of all, anticipates the current situation of the boxers based on the information delay, the execution delay and its own boxing ring simulator. After that, calculation of the Fuzzy system inputs like *Alpha*, *Distance* and *Danger* is performed.

Alpha as shown in Figure 3 represents the degree between the perpendicular line to the body of the AI boxer, and the link from the central point of a boxer to the other one. *Distance* represents the space between the boxers. Since the AI boxer tries to escape from the corners of the ring, the danger of a position should be estimated independently of the opponent's position. Hence *Danger* represents how dangerous the current position of the AI boxer is.

In the control layer, first of all, by using a script, the AI checks whether its boxer can throw a punch at the opponent or it should change its position. If its punch does not hit the opponent, the AI will call its Fuzzy system to navigate the boxer toward the best position. The main idea of Fuzzy system is dividing the problem into two sub-problems including: 1) Finding the opponent and moving toward it. 2) Adjusting the exact distance and angle for an efficient punching to the opponent and evading the opponent's punches. Herein, a Fuzzy subsystem copes with each sub-problem and a 3rd Fuzzy subsystem supervises the efficacy of each of those two subsystems.

As a direct solution to the sub-problems mentioned above, the Fuzzy system includes three Fuzzy subsystems: 1) *Near Fuzzy System*, which specifies the importance of *Distance*, the AI boxer's stamina and the distance of the AI boxer's hand to its opponent and the opponent's hand to the AI boxer. It is most effective when the AI boxer and its opponent are close to each other; 2) *Far Fuzzy System*, which specifies the importance of *Distance*, *Alpha* and *Danger*. It is most effective when the AI boxer and its opponent are far from each other; 3) *Overall Fuzzy System* which specifies the importance of the *Near Fuzzy System* verdict and the *Far Fuzzy System* verdict. It should be noted that all three Fuzzy subsystems operate during the whole decision making loops and all rules are evaluated in parallel.

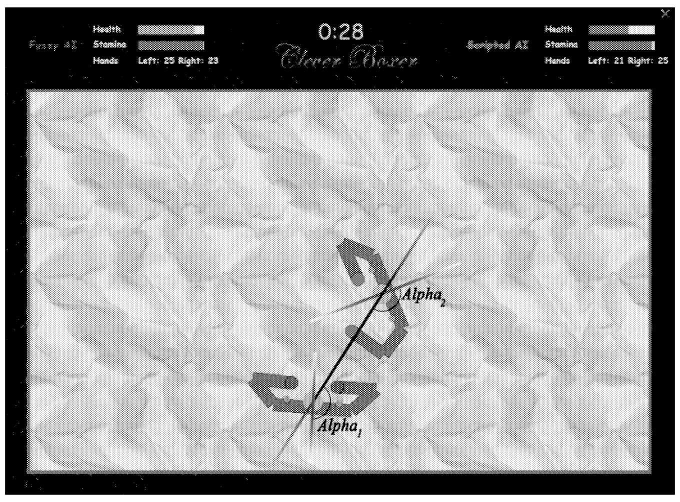


Figure 3: Displaying Alpha in a screenshot of the game

In all of the Fuzzy systems, Takagi-Sugeno Fuzzy model is employed (Takagi and Sugeno 1985). Fuzzy system input variables are codified by some simple membership functions as shown in Figure 4 and output membership functions are shaped with singletons. It is worthy to mention that the membership functions were selected intuitively at the beginning. Nevertheless after some preliminary experiments during the game development, they were tuned for a smoother game play.

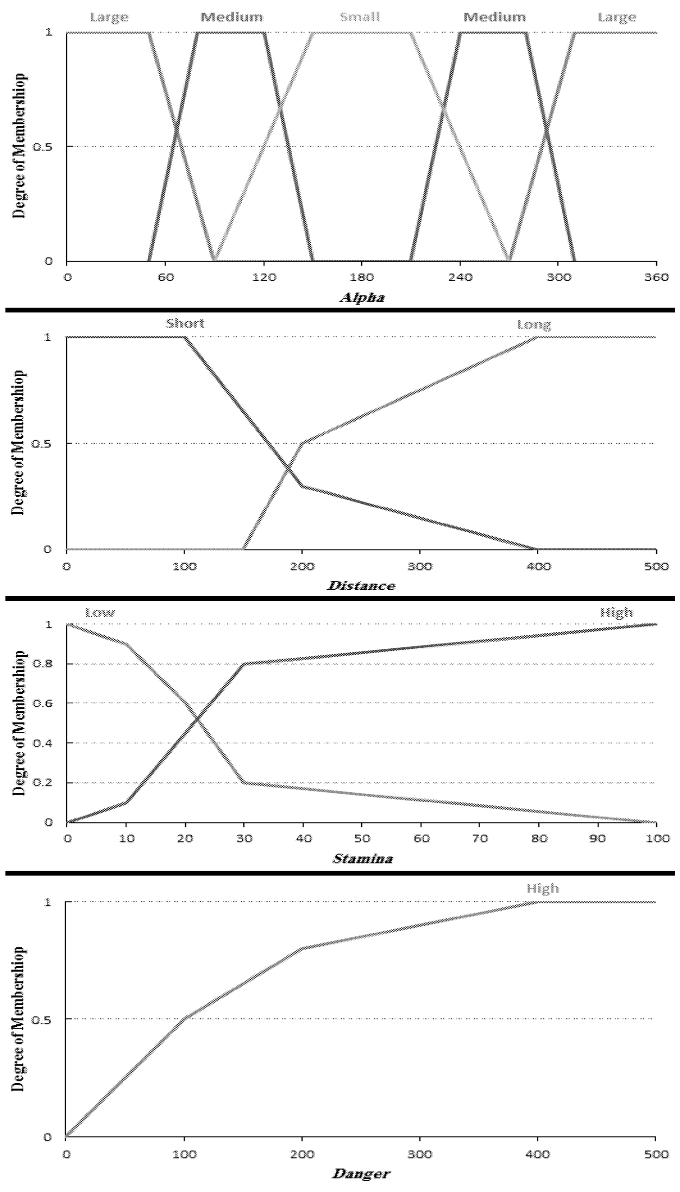


Figure 4: Fuzzy Membership Functions

Near Fuzzy System rules are shown in Table 3, where columns represent the variable *OppStamina* and rows represent the variable *MyStamina*.

Table 3: *Near Fuzzy System* Rules

OppStamia* \ *MyStamina	***Low***	***High***
Low	*W_MyHandDist* = 0.3**Attack*/5 **AND** *W_OppHandDist* = 0.3**Flee*/5 **AND** *W_Distance* = -0.3**Flee*/*Attack* **AND** *W_MyStamina* = 1	*W_MyHandDist* = 0.3*5/*Flee* **AND** *W_OppHandDist* = 1 **AND** *W_Distance* = -0.5**Flee*/5 **AND** *W_MyStamina* = 1
High	*W_MyHandDist* = 1 **AND** *W_OppHandDist* = 0.3**Attack*/5 **AND** *W_Distance* = 0.5**Attack*/5 **AND** *W_MyStamina* = 0	*W_MyHandDist* = *Attack*/5 **AND** *W_OppHandDist* = *Flee*/5 **AND** *W_Distance* = 0.3 **AND** *W_MyStamina* = 0.3

As an example in the case where both *OppStamina* and *MyStamina* are *low* the following rule will be triggered.

IF (*MyStamina* = *Low* **AND** *OppStamina* = *Low*) **THEN** (*W_MyHandDist* = 0.3**Attack*/5 **AND** *W_OppHandDist* = 0.3**Flee*/5 **AND** *W_Distance* = -0.3**Flee*/*Attack* **AND** *W_MyStamina* = 1)

Far Fuzzy System rules are shown in Table 4, where columns represent the variable *Distance* and rows represent the variable *Alpha*.

Table 4: *Far Fuzzy System* Rules

Distance* \ *Alpha	***Short***	***Long***
Low	*W_Distance* = 0.1 **AND** *W_Alpha* = 0 **AND** *W_Danger* = 1	*W_Distance* = 1 **AND** *W_Alpha* = 0 **AND** *W_Danger* = 1
Medium	*W_Distance* = 0 **AND** *W_Alpha* = 1 **AND** *W_Danger* = 0.7	
Large	*W_Distance* = 0 **AND** *W_Alpha* = 1 **AND** *W_Danger* = 0	

Finally, *Overall Fuzzy System* rules are as follows:

- **IF** (*Distance* = *Short*) **THEN** (*W_NearFuzzySystem* = 1 **AND** *W_FarFuzzySystem* = 0)
- **IF** (*Distance* = *Long*) **THEN** (*W_NearFuzzySystem* = 0 **AND** W_FarFuzzySystem = 1)
- **IF** (*Danger* = *High*) **THEN** (*W_NearFuzzySystem* = 0 **AND** W_FarFuzzySystem = 0.8)
- **IF** (*Alpha* = *Large*) **THEN** (*W_NearFuzzySystem* = 0 **AND** *W_FarFuzzySystem* = 0.7)

W_parameter denotes the weight and importance of the *parameter*, where the *parameter* can be one of the variables *MyHandDist*, *OppHandDist*, *Distance*, *MyStamina*, *Alpha* and *Danger*; A negative value of *W_Distance* states to increase the distance and a positive value states to decrease it. In addition, *W_FarFuzzySystem* and *W_NearFuzzySystem* are the degrees of truth of the statements, inferred from the outputs of *Far Fuzzy System* and *Near Fuzzy System* respectively. This is how the *Overall Fuzzy System* affects the whole decision making process. *Flee* and *Attack,* which can be configured by user between 1 and 10, specify the preference of the boxer for escaping or attacking. These two parameters can be set to 5 for a typical boxer with normal behavior. Some important notations are summarized in Table 5.

Table 5: Nomenclature

Notation	Short Definition
Alpha	Degree between two boxers
Attack	Tendency of AI boxer to attack
Danger	Danger of being near of the ring border
Distance	Distance between two boxers
Flee	Tendency of AI boxer to escape
MyHandDist	Distance of AI boxer's hand to its opponent
MyStamina	AI boxer's strength to continue fighting
OppHandDist	Distance of the opponent's hand to AI boxer
OppStamina	Opponent's strength to continue fighting

Finally, in the actuation layer, if the previous layer demands a punch, it will throw a punch based on the abilities of its hands. Otherwise it should test all possible actions on its own ring simulator during two clocks. Then a fitness factor is calculated for each action based on the weighted parameters, which have been produced by the Fuzzy system. In fact, the differences in the values of the *parameters* (such as Δ*Alpha*) during simulation are multiplied by their corresponding *W_parameter* (like *W_Alpha*) and divided by a normalizer coefficient. The sum of these resultant values determines the fitness of each action. Additionally, in the actuation layer, there are some tricks to avoid getting stuck somewhere in the ring, or persisting in just punching in a row. For instance, if the boxer retains its position near the ring edges for a while, the AI ignores the current situation of the game and moves on to increase its distance from the sides of the ring.

RESULTS

The proposed AI system has been applied to the introduced game engine. The performances of five Fuzzy systems with different configurations were assessed against a scripted AI and 10 simulations have been performed over each one. Since there is no standard method for gaging the 'believability' of game bots (Gorman et al. 2006), only the match results (wins and losses) are illustrated in Table 6.

Table 6: Game Results of the Fuzzy AI against the Scripted AI

Fuzzy AI Type	Win (Knockout)	Win (Timeout)	Lost (Knockout)	Lost (Timeout)
Balanced (*Flee*=*Attack*=5)	60%	10%	10%	20%
Defensive (*Flee* = 10, *Attack* = 1)	0%	10%	60%	30%
Offensive (*Flee* = 1, *Attack* = 10)	80%	0%	20%	0%
Fairly Defensive (*Flee* = 7, *Attack* = 3)	60%	10%	20%	10%
Fairly Offensive (*Flee* = 3, *Attack* = 7)	70%	20%	10%	0%

Based on the empirical perception about the AI agents, the most significant difference between the playing methods of the Fuzzy AI and the scripted AI is the diverse behavior of the Fuzzy AI. This matter becomes more obvious when two scripted AIs play against each other. In this case, it is seen that many repetitive situations happen in each run of the game. But when one player utilizes the Fuzzy AI, the game gets more unpredictable; In fact, the Fuzzy agent insists less on a specific action and has better adaptation to the game.

CONCLUSIONS

By using a Fuzzy system, rules and membership functions designed by a human expert, we can avoid cheating and just follow the natural laws of the game. This could be very useful for extending and making better games, i.e. when a human finds a new rule which can be useful in a game, we will be able to easily insert the new rule into the game as a Fuzzy rule.
The main benefit of the Fuzzy logic approach to game development is human-like behaviors, which guarantee believable and natural behavior (not necessarily perfect) and increase the satisfaction of human players. Another advantage of using Fuzzy logic is that, unlike scripted AI, a developer does not need to consider all possible situations in a game. Furthermore, the idea of various Fuzzy subsystems divides the state space of the problem, which makes the scaling to more demanding settings easy and could be efficiently exploited in other genres of games. As an example, in RTS games as the most AI challenging type of games, two simultaneous main concerns are consuming the resources for building different structures and training the army forces for attacking the opponent. To fulfill this purpose, two Fuzzy systems can be used for handling these two problems. In addition, a third Fuzzy system could be used for determining the priority of each of those Fuzzy systems, according to the current situation of the game.
Future enhancements are required to overcome the deliberate delay, which is applied to the AI; a simple solution would be using the probabilistic Bayesian methods in order to predict the current situation of the opponent. Another possible outcome of the *Clever Boxer* is a boxing trainer, just like a human boxing coach; The AI system watches a boxing game and then suggests how the player could improve its performance based on its own Fuzzy human-like rules.

REFERENCES

Bourg D. M. and Seeman G. 2004. *AI for Game Developers*. O'Reilly, chap. 8.

DeLooze L. L. and Viner W. R. 2009. "Fuzzy Q-Learning in a Nondeterministic Environment: Developing an Intelligent Ms. Pac-Man Agent". In *IEEE Symposium on Computational Intelligence and Games (CIG'09)*, pp. 162-169.

Doss P. 2006. "Artificial Intelligence in Computer Games". Bowie State University, M. Sci. Thesis.

Johnson D. and Wiles. J. 2001. "Computer Games with Intelligence". In *Proceedings of the 10th IEEE Conference on Fuzzy Systems* (Dec. 2-5), vol. 3, pp. 1355- 1358.

Gorman B. et al. 2006. "Believability Testing and Bayesian Imitation in Interactive Computer Games". In *Proceedings of the 9th Int. Conf. on the Simulation of Adaptive Behavior (SAB'06),* Springer, vol. LNAI.

Li Y. et al. 2004. "Fuzzy Logic in Agent-Based Game Design". In *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS '04)*, pp. 734-739.

Loiacono D. et al. 2010. "The 2009 Simulated Car Racing Championship". In *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 131-147.

Onieva E. et al. 2010. "Overtaking Opponents with Blocking Strategies Using Fuzzy Logic". In *IEEE Conference on Computational Intelligence and Games (CIG'10)*, pp123-130.

Perez D. et al. 2009. "Evolving a Fuzzy Controller for a Car Racing Competition". In *IEEE Symposium on Computational Intelligence and Games (CIG'09)*, pp. 263-270.

Rollings A. and Ernest A. 2006. *Fundamentals of Game Design.* Prentice Hall, chap. 13.

Russell S. and Norvig P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Takagi T. and Sugeno M. 1985. "Fuzzy identification of systems and its application to modeling and control". In *IEEE transaction on systems, man and cybernetics,* vol. 15, no. 1, pp. 116-132.

Tozour P. 2002. *The Perils of AI Scripting*. In Rabin S. *AI Game Programming Wisdom,* Charles River Media, pp. 541-547.

Zadeh L. 1965. "Fuzzy sets". *Inf. Control*, vol. 8, pp. 338-353.

GAME GRAPHICS

FAST, BELIEVABLE REAL-TIME RENDERING OF BURNING LOW-POLYGON OBJECTS IN VIDEO GAMES

Dhanyu Amarasinghe and Ian Parberry
Department of Computer Science & Engineering
University of North Texas
Denton, TX, USA
Email: dhanyu@gmail.com, ian@unt.edu

KEYWORDS
Hardware acceleration, deformation, procedural content generation, low-polygon modeling, CUDA, GPU.

ABSTRACT

Deformation of the low-polygon models used in video games is challenging since it is hard to maintain realism. We show how real-time mesh refinement can be used for modeling the deformation and consumption of low-polygon models under combustion while generating procedural fire. Our focus is on trading realism for computation speed so that processing power is still available for other computational tasks. Our method also allows for quick and easy LOD (level-of-detail) rendering of burning objects. We have implemented and tested our method on a relatively modest GPU (Graphics Processing Unit) using NVIDIA's CUDA (Compute Unified Device Architecture). Our experiments suggest that our method gives a believable rendering of the effects of fire while using only a small fraction of CPU and GPU resources.

INTRODUCTION

Model deformation is an essential part of maintaining the realism of physical objects in video games. While high quality graphics is an inescapable necessity for the modern video game, developers must choose detailed structure of game models carefully due to the limitations of hardware resources and processing power needed in real time rendering. One of the key features of such detailed structure is a number of polygons per model. Low-polygon models are typically used as much as possible, with their deficits hidden by a choice of pragmatic textures. As a tradeoff between quality and performance, many game developers use extremely low-polygon models for most of the flat surfaces in the game environment such as doors, windows, and walls. Since deformation of such low-polygon models while maintaining realism is quite thorny, developers commonly resort to model swapping techniques.

We consider real-time emulation of the deformation and consumption of low-polygon models due to combustion. Fire simulations may be used effectively to increase the reality of visual effects in computer animations. Real-time triangle subdivision is a useful technique, but complete subdivision of each and every model is not practical in real time. We extend the method discussed in our prior work (Amarasinghe and Parberry 2011b) to introduce a real-time refinement method that can be used in deformation and real-time rendering of burning low-polygon models while maintaining performance and realism. Our aim is to increase believability by a large amount while increasing computation load only minimally.

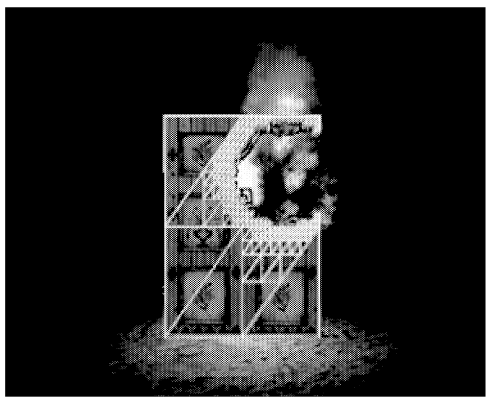


Figure 1: Procedural triangulation of a burning door.

We are able to triangulate burning low-polygon objects on-the-fly in response to real-time procedural fire in order to provide more detail where it is needed Figure 1 shows where a simple 12-triangle door is triangulated near the source of compbustion, and Figure 2 show a 12-triangle box at various stages of burning. The interested reader can visit our fire web page (Amarasinghe and Parberry 2011a) for larger color images and a video demonstration of a burning low-polygon door as shown in Figure 1.

The remainder of this paper is divided into four main sections. The first section sets the context for this paper by describing prior work. The second section describes our triangle subdivision algorithm. The third section shows that our method is particularly suited to producing models at different levels of detail for faster rendering. The fourth section describes the results of some preliminary experiments with a CUDA implementation of our algorithm.

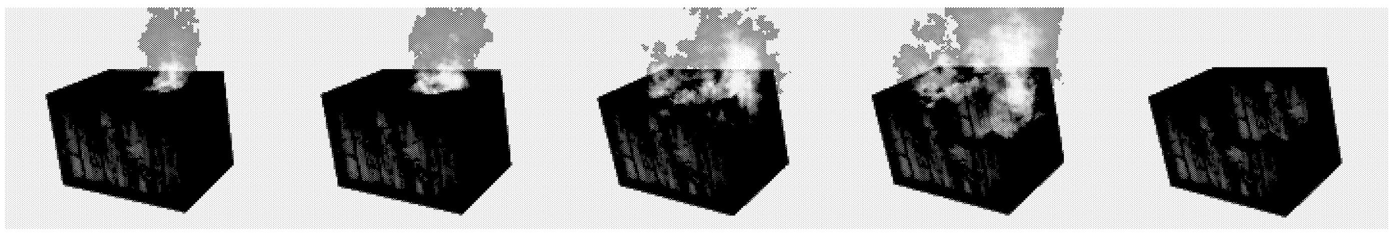


Figure 2: The consumption of a low-polygon model and the spread of procedural fire.

PRIOR WORK

In (Amarasinghe and Parberry 2011b) we describe a technique for emulating the consumption and deformation of high-polygon models due to fire. The obvious way to extend this technique to low-polygon models is to use real-time mesh refinement, subdividing triangles only when necessary.

There is a great amount of prior work on subdivision surface schemes for fast mesh refinement in real-time applications. The idea of using mesh refinement was used in (Wicke et al. 2010) to capture detailed physical behavior in simulating fractures by subdividing mesh elements. The kind of parameterizations that are optimal for remeshing are discussed in (Floater and Hormann 2005, Hormann et al. 2001). A method for adaptive mesh refinement for an expanding heat boundary is discussed in (Peyré and Cohen 2006). The papers (Guo et al. 2006), and (He et al. 2010) discuss parametric subdivision of mesh surfaces, while (Borouchaki et al. 2005) performs real time deformation by applying remeshing to selective material. Useful information about surface subdivision can be found from Kovacs and Mitchell's crease approximation method (Kovacs et al. 2009). The survey paper (Alliez et al. 2008) on remeshing of surfaces is also quite useful.

Relatively little work has been published about hardware assisted implementation of subdivision schemes. Some useful mesh refinement techniques using modern GPUs can be found in (Borouchaki et al. 2005, Boubekeur and Schlick 2005). The so-called GAMeR technique from (Schive et al. 2010) uses the GPU for adaptive mesh refinement in astrophysics. Some useful techniques for subdivision using modern GPUs can be found in (Fan and Cheng 2009) and (Settgast et al. 2004).

SUBDIVISION

Graphics Processing Units (GPUs) are no longer limited to just scene rendering. Technology such as NVIDIA's CUDA (Compute Unified Device Architecture) provides a platform for implementing general purpose computation on GPUs. However, as mentioned in (Boubekeur and Schlick 2005), there are limitations to data translation from CPU to GPU, since current graphics hardware is unable to generate more polygons than those sent through the graphics bus by the

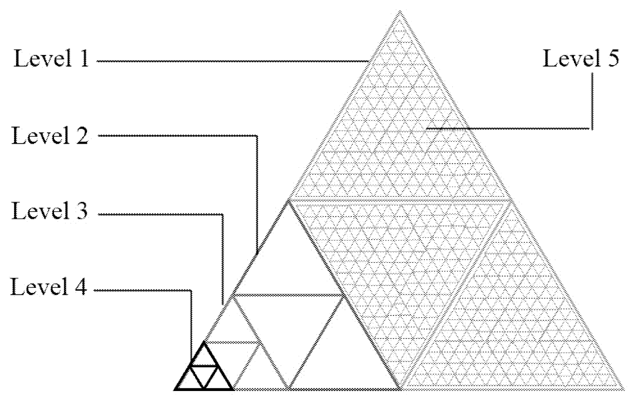


Figure 3: The Adaptive Refinement Pattern showing the level subdivisions for a single triangle.

application running on the CPU. Consequently, we have adapted their Generic Adaptive Mesh Refinement (GAMeR) technique to procedurally create additional inner vertices on-the-fly.

The remainder of this section is divided into three subsections. The first subsection discusses refinement patterns and properties. The second subsection discusses the use of barycentric points in relation to the heat boundary. The third subsection brings these concepts together in our deformation algorithm.

Refinement Patterns and Properties

Although our approach is valid for other polygonal shapes, we only consider the case of triangular meshes, since that is what is primarily used in video games. We pre-compute all of the useful refinement configurations of a single triangle using a technique called *uniform decomposition*, in which the subdivision takes place in all of the cells recursively. We use an isotropic template that divides each triangle into half for five recursive levels in depth as illustrated in Figure 3. This resulting Adaptive Refinement Pattern (ARP for short) is stored once on the GPU as a vertex buffer object.

Recall that our objective is not to subdivide each and every triangle in the object. Our aim is to subdivide only when necessary, and prior to deformation. We declare some attributes

Attribute	Values	Description
id	Integer	Track siblings/parent
SetLevel	$1, 2, 3, 4, 5$	Depth of the division
Siblings	Integer	Number of siblings
Parent	Integer	Parent id
Status	$-1, 0, 1, 2$	Status of the triangle

Table 1: ARP attributes.

for each of the subdivided triangles in Table 1. One of the important attributes in this set is the *status* of the triangle. The values -1, 0, 1, 2 represent the triangle's status as *inactive*, *initial*, *active*, and *processed* respectively. The renderer will draw only the final ARP of *active* and *processed* triangles generated by each coarse triangle.

After loading the ARP and its attributes to the vertex buffer, we need to map ARP coordinates to the corresponding coarse polygon using a displacement map similar to Figure 3. Unlike (Boubekeur and Schlick 2005), we record the final coordinate set into the GPU since we have yet to deform our vertices prior to rendering. At this point we apply ARP to the coarse polygon only if it is eligible to proceed to the next level of subdivision. This eligibility depends upon the location of the heat boundary relative to the triangle.

Barycentric Points & the Heat Boundary

The temperature of a burning object in the real world changes over both time and space. Temperature increase due to combustion influences the mechanical behavior of the object, and the thermal conductivity of the object influences the thermal response.

To speed up computation, we approximate the expansion of the heat boundary by calculating it around a single fixed point, following our heat boundary model described in (Amarasinghe and Parberry 2011b). The approximated heat boundary expansion is given by:

$$\begin{aligned} R^2 &= |\sin(\pi\Theta/\Delta r) + \sin(\pi\Theta) + \\ &\quad \psi((x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2)|, \end{aligned}$$

where $R = r + \Delta r$, the radius r incremented by Δr in each Δt time period. The angle Θ is a random value that makes the expanding heat boundary irregular in shape. The location of the heat source is (x_0, y_0, z_0). However, the value of heat index constant ψ from (Amarasinghe and Parberry 2011b), which is supposed to be a constant that depends only on the size of the coarse triangles of the model, is no longer fixed. Therefore, we let the designer set ψ depending on how many levels of subdivision are planned.

It remains to decide which coarse triangles are eligible for subdivision. This has to be a function of the expanding heat boundary. Furthermore, the subdivision has to take place prior to the deformation process. Our solution is to send a virtual heat wave through the model prior to the actual heat boundary expansion. This creates an area in addition to the three initial heat boundary areas described in Figure 3 of (Amarasinghe and Parberry 2010). Since the introduced boundary expansion takes place prior to the three original expanding boundaries (see Figure 4), we can proceed with the subdivision of qualified triangles before the deformation process begins.

Since we are using a single source heat boundary, temperature at all points will depend on the distance from the heat source at (x_0, y_0, z_0). If this point is in the middle of one of the coarse triangles, the triangle will not be eligible for subdivision until the virtual heat boundary hits one of its vertices. To avoid such issues we represent each triangle using barycentric coordinates as follows. Suppose point $P = (x, y, z)$ is given by:

$$\begin{aligned} x &= \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \\ y &= \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 \\ z &= \lambda_1 z_1 + \lambda_2 z_2 + \lambda_3 z_3, \end{aligned}$$

where λ_1,λ_2 and λ_3 are area parameters such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

We need to calculate the barycentric coordinates for non-eligible triangles only, where *eligible* triangles are those that are close to the heat boundary. The following algorithm returns `true` if coarse triangle T is eligible.

```
for each coarse triangle $T$
  if $T$ is not eligible then get barycentric point set $P$

for each barycentric point set $P$
  if $P$ is inside the heat boundary
    return true
```

Deformation

After applying ARP to the eligible triangle, we next apply deformation techniques. Although our ARP arbitrarily contains triangles of five levels in depth (see Figure 3), this number can be changed in the obvious fashion by the designer. Deformation applies only to the final level (in our case, fifth level) status *active* triangles. At this point, rendering all levels of triangles in the ARP will be costly and wasteful. Instead, we choose which triangles to render using the ARP attributes listed in Table 1.

The process can be described informally as follows. Initially, the coarse triangles of the model are considered *active* (status value 1) triangles, and all ARP triangles are initialized as *initial* (status value 0) triangles. Each subdivided triangle consists of three siblings and a parent. As our algorithm proceeds, if one of the child triangles turns *active*, then the parent will turn *processed* (status value 2) until all of its children also become status *active*. Once its children have all turned *active*, the parent triangle will change its status from

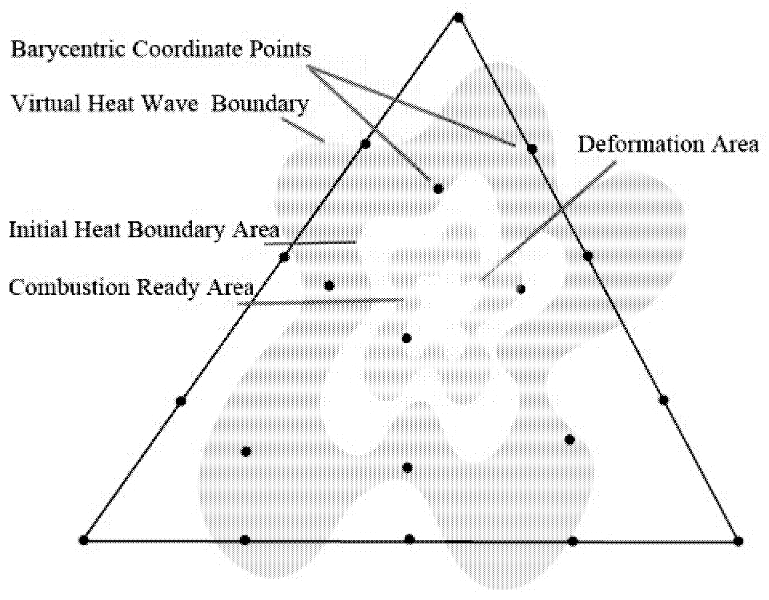

Barycentric Coordinate Points
Virtual Heat Wave Boundary
Deformation Area
Initial Heat Boundary Area
Combustion Ready Area

Figure 4: Heat boundary areas and barycentric point sets.

```
if triangle has SetLevel 5 and Status 0
  if triangle is inside the heat boundary
    Status := 1
    Status of all siblings := 1
    Status of parent := 2

if SetLevel $< 5$ and Status $> 0$
  if sibling's Status $>$ -1
    sibling Status := 1
    parent Status := 2

if all children have Status 1 and parent has Status 2
  parent Status := -1
```

Figure 5: Algorithm for computing Status.

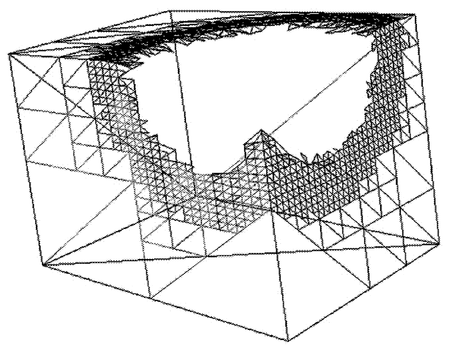


Figure 6: The refinement hierarchy and deformation applied to a 12-triangle model of a box.

processed to *inactive* (from status value 2 to -1). See Figure 5 for the details.

In our high-polygon deformation algorithm (Amarasinghe and Parberry 2011b) the displacement of each vertex depends on the surrounding vertices. Therefore, to apply proper calculation of deformation, we must let the subdivision proceed a few steps further before applying deformation to the mesh. By doing so, we are able to calculate the proper strength factors within the deforming triangles properly. Figure 6 illustrates the refinement hierarchy and deformation applied to a low-polygon model of a box.

LEVEL OF DETAIL

In a game environment, objects located far from the viewer need not be rendered in as much fine detail as those close up. A significant speed-up can be obtained by having models stored at various *levels of detail* (abbreviated LOD) ranging from, for example, hundreds of triangles for objects in the far distance to tens of thousands for close-up objects. These variants of the model are usually created by the artist, although procedural methods do exist.

Our algorithm allows us to implement LOD for burning objects by controlling the level of adaptive refinement of the coarse mesh triangles. We calculate the distance between object and the player in the CPU and pass it to the GPU as a parameter. Level adjustment is decided and passed to the appropriate ARP before rendering.

For a solid object the level of refinement is directly proportional to the distance. However, surface removal and deformation of a burning object makes it slightly more challenging to maintain a smooth transition between level swaps. Define

Polygon Count	**Fully Subdivided**	**Our Method**	**Speed-up Factor**
10k	84fps	165fps	1.96
15k	76fps	159fps	2.09
20k	63fps	153fps	2.43
50k	48fps	60fps	1.25

Table 2: Frame rate of fully subdivided model versus our approach.

the *burn level* of a model as the number of triangles of the model that have been consumed by fire as in (Amarasinghe and Parberry 2011b). We then use the following algorithm to determine whether to render triangle T: Show T if and only if the number of children of T with higher burn level than T is ≥ 2, and SetLevel ≥ 5. Figure 7 shows our burning box at different LODs.

EXPERIMENTS

The images of burning objects shown here and in (Amarasinghe and Parberry 2011a) are screenshots from a CUDA implementation of our algorithm applied to a model with 12 triangles. The flames are generated using 2000 fire particles and 500 smoke particles. The advantage of such a system is clear when comparing the resources required to deform a completely subdivided model versus deforming a low-polygon model using our method. Table 2 shows the frame rates of the animation when our algorithm is implemented in CUDA on relatively modest hardware; An Intel®Core^{TM}2 Duo CPU P8400 @ 2.26GHz processor with an NVidia GeForce 9800 GTS graphics card. This performance will of course be much better on the current generation of graphics hardware, but that is not our aim. Our aim is to provide detail sufficient to trigger willing suspension of disbelief at a relatively low cost in computation load.

The outcome of these experiments shows that our method results in doubling the frame rate. Therefore, we believe this approach is a better alternative than subdividing the complete model when it comes to deforming low-polygon models.

CONCLUSION AND FURTHER WORK

We have proposed a method for the real-time deformation and consumption of a low-polygon model during combustion by procedurally generated fire. By doing so, we have extended our work in (Amarasinghe and Parberry 2011b) to low-polygon models. We have performed simulation of real-time deformation and consumption of any model regardless of the size of the triangles. We have focused on the performance with a reasonable amount of realism sufficient to trigger willing suspense of disbelief in the game player. Our simulations have performed well on a model with low-polygon count and large triangles. We intend to investigate the extension of our method to solid models, and to investgate a better approximation to heat boundary expansion.

REFERENCES

Alliez P.; Ucelli G.; Gotsman C.; and Attene M., 2008. *Recent advances in remeshing of surfaces*. *Shape Analysis and Structuring*, 53–82.

Amarasinghe D. and Parberry I., 2010. *Towards Fast, Believable Real-time Rendering of Burning Objects in Video Games*. Tech. Rep. LARC–2010–04, Laboratory for Recreational Computing, Dept. of Computer Science & Engineering, Univ. of North Texas.

Amarasinghe D. and Parberry I., 2011a. *Fire Reloaded*. URL `http://larc.unt.edu/ian/research/fire2/`.

Amarasinghe D. and Parberry I., 2011b. *Towards Fast, Believable Real-time Rendering of Burning Objects in Video Games*. In *Proceedings of the 6th Annual International Conference on the Foundations of Digital Games*. 256–258.

Borouchaki H.; Laug P.; Cherouat A.; and Saanouni K., 2005. *Adaptive remeshing in large plastic strain with damage*. *International Journal for Numerical Methods in Engineering*, 63, no. 1, 1–36.

Boubekeur T. and Schlick C., 2005. *Generic mesh refinement on GPU*. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. ACM, 99–104.

Fan F. and Cheng F., 2009. *GPU Supported Patch-Based Tessellation for Dual Subdivision*. In *2009 Sixth International Conference on Computer Graphics, Imaging and Visualization*. IEEE, 5–10.

Floater M. and Hormann K., 2005. *Surface parameterization: A tutorial and survey*. *Advances in Multiresolution for Geometric Modelling*, 157–186.

Guo X.; Li X.; Bao Y.; Gu X.; and Qin H., 2006. *Meshless thin-shell simulation based on global conformal parameterization*. *IEEE Transactions on Visualization and Computer Graphics*, 375–385.

He L.; Schaefer S.; and Hormann K., 2010. *Parameterizing subdivision surfaces*. *ACM Transactions on Graphics*, 29, no. 4, 1–6.

Hormann K.; Labsik U.; and Greiner G., 2001. *Remeshing triangulated surfaces with optimal parameterizations*. *Computer-Aided Design*, 33, no. 11, 779–788.

Kovacs D.; Mitchell J.; Drone S.; and Zorin D., 2009. *Real-time creased approximate subdivision surfaces*. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. ACM, 155–160.

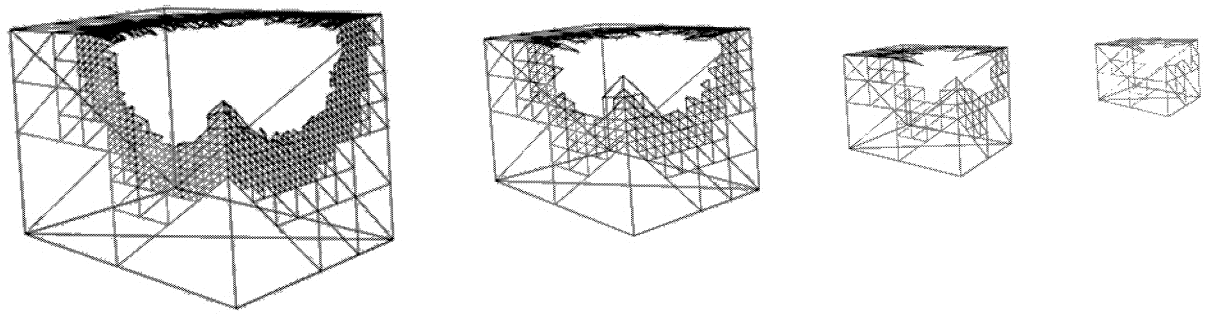


Figure 7: Levels of detail for a burning object provided by our method.

Peyré G. and Cohen L., 2006. *Geodesic remeshing using front propagation*. *International Journal of Computer Vision*, 69, no. 1, 145–156.

Schive H.; Tsai Y.; and Chiueh T., 2010. *GAMER: A graphic processing unit accelerated adaptive-mesh-refinement code for astrophysics*. *The Astrophysical Journal Supplement Series*, 186, 457.

Settgast V.; Müller K.; Fünfzig C.; and Fellner D., 2004. *Adaptive tesselation of subdivision surfaces*. *Computers & Graphics*, 28, no. 1, 73–78.

Wicke M.; Ritchie D.; Klingner B.; Burke S.; Shewchuk J.; and O'Brien J., 2010. *Dynamic local remeshing for elastoplastic simulation*. *ACM Transactions on Graphics*, 29, no. 4, 1–11.

BIOGRAPHY

DHANYU AMARASINGHE is a native of Sri Lanka. He is currently a PhD student in the Department of Computer Science and Engineering at the University of North Texas. His research interests include graphics for game development, particularly the real-time deformation and consumption of virtual objects by procedural fire. He can be contacted at `dhanyu@gmail.com`. His home page is `http://dhanyu.com/`.

IAN PARBERRY was born in London, England and emigrated as a child with his parents to Brisbane, Australia. After obtaining his undergraduate degree there from the University of Queensland he returned to England for a PhD from the University of Warwick. He has worked in academia in the US ever since. He is currently a full Professor in the Department of Computer Science and Engineering at the University of North Texas where he recently stepped down from a 2-year term as Interim Department Chair. A pioneer of the academic study of game development since 1993, his undergraduate game development program was ranked in the top 50 out of 500 in North America by The Princeton Review in 2010. He is on the Editorial Boards of the *Journal of Game Design and Development Education*, *IEEE Transactions on Computational Intelligence and AI in Games*, and *Entertainment Computing*, and he serves as the Secretary of the Society for the Advancement of the Science of Digital Games, which organizes the Annual Foundations of Digital Games conference. He is the author of 6 books and over 80 articles over 30 years' experience in academic research and education. His *h*-index is 18 and his Erdös number is 3. He can be contacted at `ian@unt.edu` or on Facebook. His home page is `http://larc.unt.edu/ian`.

VERY FAST REAL-TIME OCEAN WAVE FOAM RENDERING USING HALFTONING

Mary Yingst
Dept. of Computer
Science & Engineering
University of North Texas
Denton, TX, USA
maryyingst@my.unt.edu

Jennifer R. Alford
Digital Teapot, Inc.
Fort Worth, TX, USA
gralford@acm.org

Ian Parberry
Dept. of Computer
Science & Engineering
University of North Texas
Denton, TX, USA
ian@unt.edu

KEYWORDS

Physics and Simulation, Design

ABSTRACT

We introduce an efficient method for emulating sea foam dissipation suitable for use in real-time interactive environments such as video games. By using a precomputed dither array with controlled spectral characteristics adopted from halftone research as a control mechanism in the pixel shader, we can animate the appearance of foam bubbles popping in a random manner while allowing them to clump naturally.

INTRODUCTION

Real-time animation and rendering of ocean waves is often seen in video games, and adding foam to the waves lends an added level of realism. We describe a fast and effective method for rendering ocean wave foam by augmenting traditional texture based foam saturation methods with techniques from halftoning.

Takahashi et al. (2003) and Thürey et al. (2007) represent foam as a particle system. Although this is visually pleasing, it is computationally intensive. In large scale environments such as the ocean it is more practical to use faster texture based methods. Many methods of rendering foam rely on applying a texture of foam to the water surface. These methods apply a texture using a foam *saturation*, or *density* value to represent transparency of the texture which is applied to a mesh representing the water's surface (see, for example, Jensen and Golias (2001), Jeschke et al. (2003), and Kryachko (2005)). Li et al. (2008) similarly apply a foam color according to its density.

Real ocean foam consists of bubbles clumped together by surface tension on the surface of the water. Foam does not simply fade or become transparent as the bubbles dissipate. Traditional methods of foam generation ignore the active nature of foam density where bubbles pop over time. Since surface bubbles are either present or not in an area of water, this binary nature lends itself to the use of halftoning, a process used to reproduce images using patterns of black dots. Our use of halftoning with a saturation function that changes over time causes bubbles to appear to pop.

The remainder of this note is divided into five sections. First we give a high-level overview of our approach. Then we review in more depth our choice of foam saturation function, our use of a halftoning mask generated using methods from the halftoning literature, and how we apply that mask in a pixel shader. Finally we conclude with a discussion of our results.

OVERVIEW OF OUR APPROACH

To generate foam on the surface of the water using a foam saturation function, we must create the water surface as a mesh. Each location on the water's surface has a calculable saturation value using this function. The function must vary over time for the foam to animate and become more and less dense as waves pass and change. In figure 1 we see that by applying halftoning methods to a saturation function, we take an otherwise smooth area of the function and create the randomness expected when foam generates and dissipates.

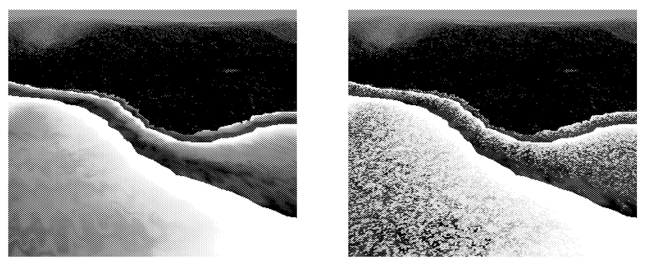


Figure 1: By replacing the application of a foam texture with a white tone we see that applying our method creates randomness on the right in the otherwise smooth saturation results pictured on the left.

Halftone masks, or *dither arrays*, are arrays of values that have a one-to-one correspondence with pixels in an image, or in our application, a texture. Each value of the halftone mask is used as a threshold against the corresponding texture pixel to produce a binary output image that indicates, at each pixel position, whether the texture falls above or below the threshold. This process is commonly referred to as *thresholding*. Halftone masks are characterized by the binary pattern that results when thresholded against a constant image, or texture. Choosing threshold value values at each mask position is non-trivial. Ulichney (1987) provides a classic study of mask design and describes widely used metrics, based on the Fourier Transform, to characterize masks by their radially averaged power spectrum (RAPS), a measure of energy at different frequency bands, and anisotropy, a measure of radial symmetry. While halftoning can be accomplished with a variety of computational methods, we restrict ourselves to the use of masks because, as point operations, they are computationally efficient and naturally suited to pixel shader operations.

We depart from the traditional use of halftoning in printing and image display, which seeks to reduce visually objectionable noise in image reproduction, and instead we use a halftone mask to add noise. We draw on recent work in halftone mask design by observing that it is possible to design masks to produce lumping binary patterns which are reminiscent of the clumping of sea foam. We also observe that the binary nature of the threshold output is well-suited to simulate foam bubble popping when the mask is fixed per frame but the underlying image is not. In this work, we present a novel way to use halftone masks in conjunction with a saturation function and a texture to simulate foam and the popping behavior of foam. Further, we observe that the difference between the threshold value and an image or a texture provides a magnitude at each pixel position that we use as a transparency value for additional realism.

We use halftone masks that have been generated using a symmetric Gaussian function to filter white noise as described in Alford and Sheppard (2010). Gaussian filtering applies a two-dimensional Gaussian function to an image. σ is a value in the Gaussian function that denotes the width of the curve in the function; as σ increases, the width of the curve increases.

We simulate the effect of foam bubbles popping by finding the saturation of foam on the water's surface and applying a precomputed halftone mask to it. We use a modified version of the vertex shader outlined in a paper by Van Drasek III et al. (2010) to create parametric waves upon which to apply our foam. The next two sections will describe the saturation function and the halftone mask in more detail.

THE SATURATION FUNCTION

Kryachko (2005) uses the following foam saturation function which is dependent on ocean height. H_0 is base height, H is height, and $H_{\max}$ is height where foam is maximum.

$$f(x) = \frac{H - H_0}{H_{\max} - H_0}$$

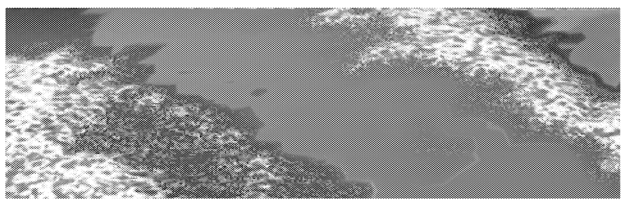


Figure 2: Foam with Kryachko's saturation function.

Although Kryachko's function achieves somewhat attractive results (see Figure 2 for example), the function results in a symmetric foam distribution, whereas we wish to model foam that is created by turbulence at the front of the wave and fades away behind it. Knowing the target foam density along the wave shape, we chose to apply $e^{\tan(x)}$ to the same vector and frequency used to determine wave shape.

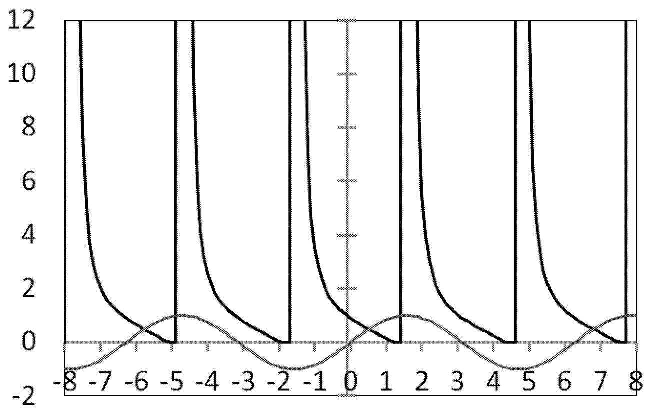


Figure 3: $e^{\tan(x)}, \sin(x)$

We use the following formulae from Van Drasek III et al. (2010) for the height y of the wave:

$$\begin{aligned} y &= A((\sin(\theta(x,z)) + 1)/2)^K \\ \theta(\vec{v}) &= (\vec{v} \cdot \vec{k})2\pi/\lambda_{adj} + \phi t \\ \phi &= 2s\pi/\lambda, \end{aligned}$$

and so we use $\theta(\vec{v})$ to also generate the periodic function.

$$f(\vec{v}) = e^{-\tan((\vec{v}\cdot\vec{k})2\pi/\lambda_{adj} + \phi t))},$$

where $\vec{v} = (x, z)$ is position, $\vec{k}$ is the wave direction, s is the speed of the wave, t is time, K is wave slope, A is wave amplitude, λ_{adj} is wavelength adjusted for ocean depth, and λ is original wavelength.

Since we are overlaying this function on the sine function that determines wave shape, we need to modify the formula slightly to align the foam with the waves. In Figure 3 we see that $e^{\tan(x)}$ is twice as frequent as $\sin(x)$, so we divide $\theta(x, z)$ by 2. Also to align the highest part of our function with the front part of the sine wave we add $\pi/2$. Our final formula is as follows, and gives a attractive saturation of foam starting at the wave front and fading behind it.

$$f(\vec{v}) = (e^{-\tan((\vec{v}\cdot k)\pi/\lambda_{adj}+\phi t/2)+\pi/2)})/C,$$

where C is a user defined constant that governs the intensity of the foam. (We use $C = 4$ for convenience, but this value may be tuned by the designer.)

Saturation is computed as follows. Adjwavelength, and phaseC are calculated in the vertex shader and the values are interpolated for use in the pixel shader.

```
float getmysaturation(float2
   wavedirection, float2 xzposition,
   float Adjwavelength, float phaseC)
{
   float result =
      dot(wavedirection, xzposition)
      *6.28f/Adjwavelength;
   result = result + phaseC*gTimeNow;
   result = pow(2.718f,-1.0f*
   tan((result/2.0f)+ 1.07f))/4.0f;
   return result;
}
```

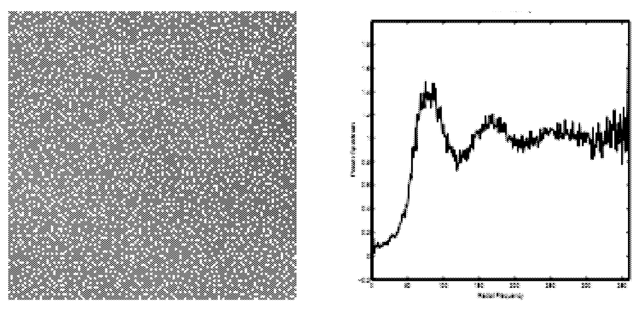
To pass values from the vertex shader we simply define an extra variable in the vertex output with a TEXCOORD semantic. Then the vertex shader sets the required values as follows:

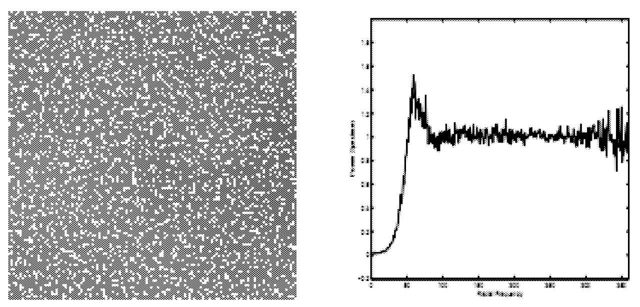
```
struct VertexOutput
{
   ...
   float4 impVars : TEXCOORD4;
}

VertexOutput VS(...)
{
   VertexOutput OUT = (VertexOutput)0;
   ...
   OUT.impVars[1]=adjustedWavelength;
   OUT.impVars[0]=phaseConstant;
   OUT.impVars[2] = Po[0]; //xposition
   OUT.impVars[3] = Po[2]; //zposition
}

float4 PS(VertexOutput IN) : COLOR
{
   float saturated=getmysaturation
      (direction, float2(IN.impVars[2],
      IN.impVars[3]), IN.impVars[1],
      IN.impVars[0]);
   ...
}
```

THE HALFTONE MASK

We use a halftone mask to threshold the saturation function to create dissipation through bubble popping. As saturation decreases over time at a specific location, the value will approach and pass the threshold used in our mask. While the saturation value is above the threshold, the foam will be present, but as time passes and the value decreases, eventually the foam will *pop* and dissappear. Since bubbles in foam clump, we must choose a halftone mask that produces clumps in the resulting dot patterns. Clumpiness, or clustering, can be seen in how close together some of the foam is while in other areas there are gaps.

Alford and Sheppard (2010) show a variety of halftone masks created using radially symmetric Gaussian filters. We used their masks created using filters having σ ranging from 1.5 to 24 to produce the images in Figure 4 column 1. In Figure 4 we can see that the higher the σ, the closer together some of the dots are. By analyzing the RAPS we see that as σ increases, first oscillation is dampened in the high frequencies, then the values of the high frequency region is greatly reduced (Alford and Sheppard 2010). The results of this can be seen in the increased clustering and clumping behavior of the dot patterns. We found $\sigma = 24$ gives adequate visual clusters of foam.


(a) $\sigma = 1.5$


(b) $\sigma = 6$

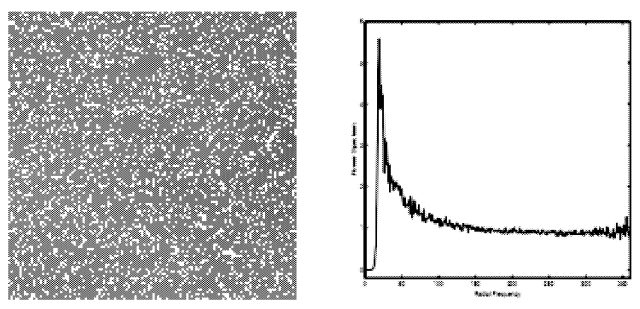

(c) $\sigma = 24$

Figure 4: Halftone masks created by Gaussian filters having σ ranging from 1.5 to 24, with corresponding RAPS (images courtesy Alford and Sheppard (2010)).

APPLYING THE MASK

To create the halftoned saturation function $h(u, v)$ where u, v are texture coordinates and $h(u, v)$ is a `float4` RGBA color value at that position, we first create a texture to contain the mask information so that the data can be imported into the pixel shader. Given a 512×512 halftone mask, a 512×512 pixel texture is generated. This texture, when tiled across the surface of the water, has a corresponding u, v texture coordinate for each $\vec{v} = (x, z)$ position on the water. The mask value $m(u, v)$ can then be used to threshold the saturation function $f(x, z)$ as follows:

$$h(u,v) = \begin{cases} (0,0,0,1) & \text{if } f(x,z) \leq m(u,v) \\ (1,1,1,1) & \text{if } f(x,z) > m(u,v) \end{cases} \tag{1}$$

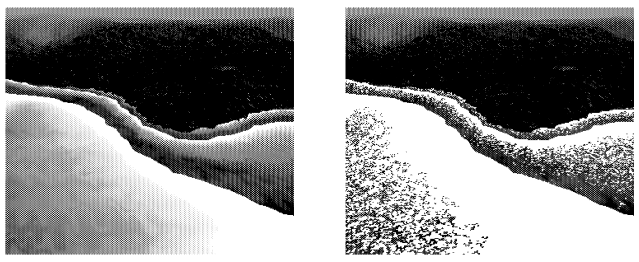


Figure 5: Applying Equation 1 to the saturation function at left gives the image at right.

We can then create a fading halftoned saturation function, $g(u, v)$, so the dots fade before they pop. We do this by taking the difference between saturation and mask number. Figure 1 shows the results of applying halftoning with fading to the saturation function. For all $g(u, v)$, $\alpha = 1$.

$$\begin{aligned} g(u,v).rgb \;=\; & \text{clamp}(f(x,z)/2 - m(u/v), 0, 1) * \\ & h(u,v).rgb \end{aligned} \tag{2}$$

Finally we apply $t(u, v)$, the foam texture to generate the final halftoned, textured, and faded image $j(u, v)$. For all $j(u, v)$, $\alpha = 1$.

$$j(u,v).rgb = \begin{cases} (0,0,0) & \text{if } f(x,z) \leq m(u,v) \\ g(u,v)* & \\ \quad t(u,v).rgb & \text{if } f(x,z) > m(u,v) \end{cases} \tag{3}$$

Given a sampler for the halftone mask texture, MaskSampler; a sampler for the foam texture, SAMP_FoamTexture; and a sampler for the water surface texture, SAMP_WaterTexture; the following code finds the resulting color for the water's surface. The higher TEXscale or MASKscale is, the smaller the tiled texture will appear. A value of 400 for MASKscale gives suitably sized dots when using a 512×512 pixel mask.

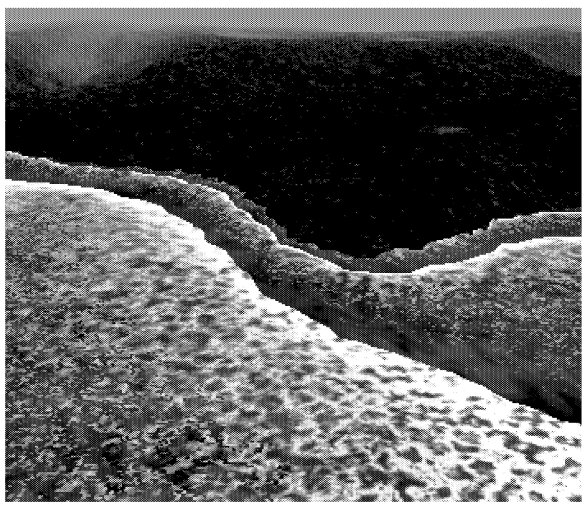
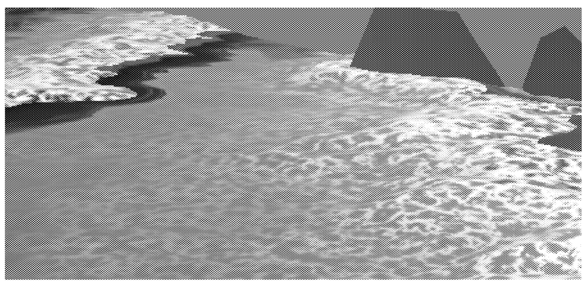


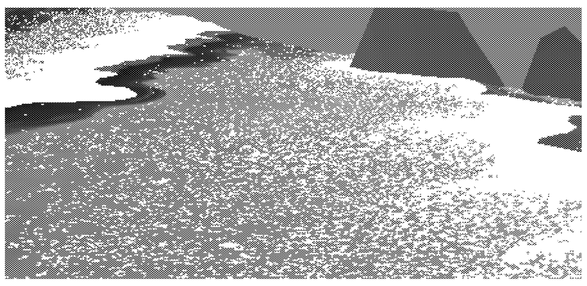
Figure 6: Coastline image using our new halftoning method, Equation 3.

```
//get water and foam texture color
float4 textureSamp = tex2D(
  SAMP_WaterTexture,
  IN.TexCoord1*TEXscale);
float4 foamSamp = tex2D(
  SAMP_FoamTexture,
  IN.TexCoord1*TEXscale);


//get the threshold from the mask
float masknumber=(tex2Dlod(
  MaskSampler, float4(IN.TexCoord1.xy
  *MASKscale,0,0)));

//threshold the saturation value
if(!((saturated)>(masknumber))){
        foamSamp[0] = 0;
        foamSamp[1] = 0;
        foamSamp[2] = 0;
}


//find the value for fading the foam
float difference = clamp(saturated
  -     masknumber, 0.15f, 3.0f);

//get the final foam value
foamSamp = difference * foamSamp;

//add the value to the water texture
//and clamp to a valid color
float4 result =clamp((textureSamp+
  foamSamp),0,1);
result[3] = 1.0f;
```

RESULTS

Figure 7(a) shows the traditional method of fading a foam texture according to a saturation function, similar to Kryachko (2005). Figure 7(b) shows the saturation halftoned using Equation 1 and no other functions applied. This method shows a realistic popping effect, but the foam is too harsh and white. Figure 7(c) shows our halftoning method in combination with a foam texture using Equation 3.


(a) Using a foam texture.


(b) Using a halftone mask to determine foam location.

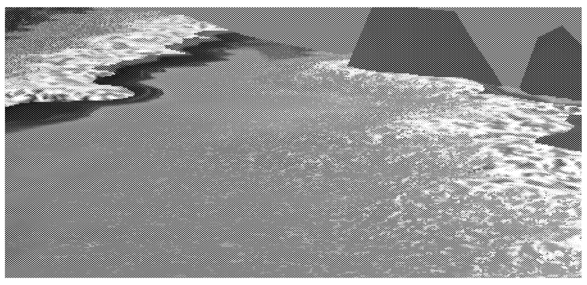

(c) Using a halftone mask with a foam texture.

Figure 7: The results of using 3 different methods with the same settings (heightmap, wave speed, direction, and amplitude).

We performed some experiments to obtain a preliminary benchmark for the extra computation load required by our new halftoning technique (Figure 7(c)) to the traditional texturing technique (Figure 7(a)). We ran both algorithms for five minutes using NVidia Composer, using FRAPS to measure average frames-per-second. The scene rendered in all

Graphics Card	Textured	Halftoned
NVidia 8800	65.5fps	65.0fps
NVidia GeForce GT320m	80.4fps	78.2fps
Intel HD Graphics 3000	85.0fps	84.5fps

Table 1: Comparison of rendering frame rates in frames per second (fps).

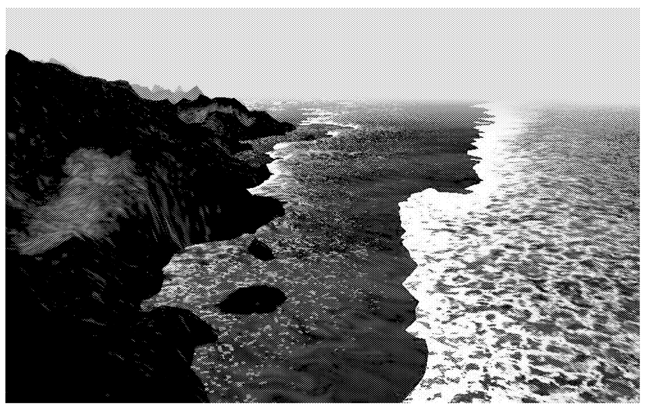


Figure 8: Scene used for measuring frame rates.

experiments is shown in Figure 8. The results are shown in Table 1. We conclude that the extra load on the video appears to be less that 3% higher than traditional texture-fading techniques, which is negligible.

Still pictures such as shown in Figure 7 and Figure 8 do not adequately capture the full effect of our algorithm. Figure 9 shows how foam bubbles fade and pop over time in the wake of each wave. This can be seen to best advantage in an animation such as the one we have placed online at Yingst et al. (2011).

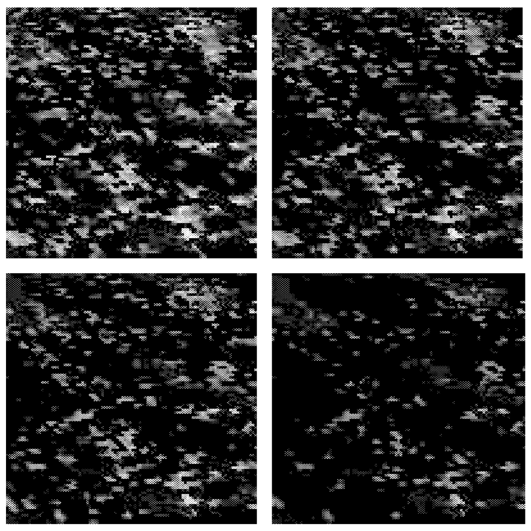


Figure 9: Close up view of foam bubbles fading and popping over time.

CONCLUSION AND FURTHER WORK

Not only does our halftoning technique achieve our goal of simulating foam dissipation in a real-time environment, but it also can be applied with little additional cost to traditional texture based methods that obtain foam saturation at the water's surface. The saturation function used must vary over time for the bubble popping effect to occur using the halftoning method.

Our method currently produces pixelation at close range to the camera. One method for remedying this would be a second pass of a pixel shader to smooth the edges of the generated texture, which we leave as future work.

REFERENCES

Alford J.R. and Sheppard D.G., 2010. *Approximating Poisson Disk Distributions by Means of a Stochastic Dither Array*. In *EG UK Theory and Practice of Computer Graphics*.

Jensen L.S. and Golias R., 2001. *Deep-Water Animation and Rendering*. URL `www.gamasutra.com/gdce/2001/jensen/jensen_01.htm`. Presented at Game Developers Conference, Europe.

Jeschke S.; Birkholz H.; and Schmann H., 2003. *A Procedural Model for Interactive Animation of Breaking Ocean Waves*. In *Proceedings of WSCG 2003*. WSCG.

Kryachko Y., 2005. *Using Vertex Texture Displacement for Realistic Water Rendering. GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*.

Li Y.; Jin Y.; Yin Y.; and Shen H., 2008. *Simulation of shallow-water waves in coastal region for marine simulator*. In *Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. ACM, 15:1–15:5.

Takahashi T.; Fujii H.; Kunimatsu A.; Hiwada K.; Saito T.; Tanaka K.; and Ueki H., 2003. *Realistic Animation of Fluid with Splash and Foam. Computer Graphics Forum*, 22, no. 3, 391–400.

Thürey N.; Sadlo F.; Schirm S.; Müller-Fischer M.; and Gross M., 2007. *Real-time simulations of bubbles and foam within a shallow water framework*. In *Proceedings of the 2007 ACM SIGGRAPH Eurographics Symposium on Computer Animation*. Eurographics Association.

Ulichney R., 1987. *Digital Halftoning*. Cambridge, Mass: The MIT Press.

Van Drasek III J.; Bookout D.; and Lake A., 2010. *Real-Time Parametric Shallow Wave Simulation*. URL `http://software.intel.com/sites/billboard/article-archive/real-time-parametric/`.

Yingst M.; Alford J.R.; and Parberry I., 2011. *Sea Foam*. URL `http://larc.unt.edu/ian/research/seafoam/`.

BIOGRAPHY

MARY YINGST was born and raised in Texas. She received her BS in computer science in 2007, and is a recent graduate of the University of North Texas, College of Engineering with a MS in Computer Science. She has participated in the game development program at UNT for several years, fostering her research interests which include graphics for game development and real-time simulation. Her Erdös number is 4. Her home page is `http://maryingst.net`.

JENNIFER (GINGER) ALFORD is Director of Computer Sciences at Trinity Valley School and President of Digital Teapot, Inc, (www.digitalteapot.org). She holds a PhD in electrical and computer engineering from the University of Iowa with a specialization in image processing and particular expertise in halftoning. Her 25 years experience in image processing and computer graphics include industrial research and software development, serving as an technical consultant and expert witness for intellectual property attorneys, college and graduate level teaching, and several academic publications. Dedicated to research and education, she has partnered with the Laboratory for Recreational Computing at the University of North Texas as a Research Associate.

IAN PARBERRY was born in London, England and emigrated as a child with his parents to Brisbane, Australia. After obtaining his undergraduate degree there from the University of Queensland he returned to England for a PhD from the University of Warwick. He has worked in academia in the US ever since. He is currently a full Professor in the Department of Computer Science and Engineering at the University of North Texas where he recently stepped down from a 2-year term as Interim Department Chair. A pioneer of the academic study of game development since 1993, his undergraduate game development program was ranked in the top 50 out of 500 in North America by The Princeton Review in 2010. He is on the Editorial Boards of the *Journal of Game Design and Development Education*, *IEEE Transactions on Computational Intelligence and AI in Games*, and *Entertainment Computing*, and he serves as the Secretary of the Society for the Advancement of the Science of Digital Games, which organizes the Annual Foundations of Digital Games conference. He is the author of 6 books and over 80 articles over 30 years' experience in academic research and education. His *h*-index is 18 and his Erdös number is 3. He can be contacted at `ian@unt.edu` or on Facebook. His home page is `http://larc.unt.edu/ian`.

MOTION IN GAMING

Experimental Soccer Robot Identification Using Light-Emitting Diodes

Eliška Ochodková, Tomáš Kocyan, Václav Svatoň, Jan Martinovič
FEECS, VSB – Technical University of Ostrava
Czech Republic
E-mail: {eliska.ochodkova,tomas.kocyan,vaclav.svaton,jan.martinovic}@vsb.cz

KEYWORDS
Robot Soccer, LED Diodes, Vision System, Lego Robots

ABSTRACT

Image analysis is one of the key elements of the architecture of each robot soccer game. Today used camera systems differ mainly in a number of used cameras, a place of their location or method used in image analysis. One thing that all these systems have in common is sensitivity to light. When lighting conditions are changed, they must often perform a new calibration of the vision system, or it is necessary to use adaptive vision system that performs the calibration automatically. This article presents a new method of marking robots based on LED diodes and familiarization with the advantages and disadvantages that this new marking bring so that the system was the least dependent on the change of lighting.

INTRODUCTION

In a game of robot soccer, one of the most important aspects of game play is image analysis. Its task is to detect current position of robots and the ball, together with providing as much additional information as possible, for example the robot's orientation or its role in the game. Image analysis deals with the problem of identification of these key features on the game field, which include the method of distinguishing the opponent's robots from your own, detecting the ball or the game field itself. This article includes a brief overview of nowadays used camera system types and methods of robot marking, which are used in robot soccer. Furthermore, an own robot marking concept is introduced and the motivation leading to the creation of it is presented. Probably the best-known representative of robot soccer is an international organization called Robocup (http://small-size.informatik.uni bremen.de (2011)), which organisms robot soccer tournaments in several categories. For example, in the SSL (Small Size League) category, there compete small, square-shaped robots of maximum height of 15 cm. colour segmentation, a method using a reference table with colour codes, is used for their identification. The colour code is simply a colourful piece of paper attached to the top of the robot. It clearly determines its team membership. Based on the order or the organization of colours, it is possible to determine the robot's orientation or position, or whether it is a goalkeeper, a defender, or a forward. The game field itself is black, which helps the camera system to easily detect the colour-marked robot and the ball, which is orange. Until 2010, the teams in the Robocup SSL category were allowed to set up their own camera system (Zickler et al. (2009)). That brought many problems like time-consuming camera adjusting and their calibration, while the single camera systems did not vary at all. The executive commission therefore made a decision about the use of common camera system called SSL-Vision (http://code.google.com/p/ssl vision (2011)). The image is scanned by one common camera system, then processed by image server and the data acquired are distributed to both competing teams. The next big name in the field of robot soccer is FIRA (The Federation of international Robot-soccer Associations, (http://www.fira.net (2011))). FIRA, as well as Robocup, organizes soccer tournaments divided into several categories, from which NaroSot resembles the SSL category very much. Each competing team has its own camera system above the game field. However the camera system's position is limited to the central or the team's own part of the game field. Still, this method brings the same problems as in Robocup.

CAMERAS

Important aspects, when it comes to image analysis, are the number and the location of cameras themselves, which scan the game field. The output of each camera is an image of the game field, which is also an input into yet-mentioned image analysis. There are several options, how to place a camera or a group of them. A typical example (as in Robocup and FIRA competitions) is a single camera placed above the field in the middle of it; the image from it is distributed to both teams. Another option is using one or more cameras that are attached above the team's own half. One of the less-used ways is for example scanning from the side of the field, when the problems with acquired image perspective have to be solved and the height of scanned objects needs to be taken into account. The methods listed above are used with static camera systems, whose position does not change during the game. However that is not a condition in dynamic camera systems class when the camera position can change during the game, but the image analysis has to consider camera position and the height of objects scanned, as in the case of dealing with perspective. These types of camera systems are used mainly with centralized control, when we get information about location of all objects currently on the game field. Based on the information, it is decided what each robot is supposed to do. The other type of control, autonomous control (Vctor M. Gmez and Matelln (2003)), does not use a central camera system. Instead, a camera is attached to each robot, which makes decisions on its own based on the information provided by camera. Further ahead, we will focus mainly on centralized control, for which our concept is made.

LIGHT
What all of the methods mentioned have in common is their sensitivity to light. The aim of image analysis is to detect the robots, the ball, or the game field in every image. These objects are coloured differently from each other. Commonly used methods of image analysis include colour segmentation (Gerd Mayer and Kraetzschmar (2004)), detection of so called blobs (Sridharan and Stone (2005)) or image thresholding (Gregor Klanar and Karba (2001)). As the rules of competitions like Robocup or FIRA say very clearly what light conditions should be on the game field (Anders J. Johannson and Balkenius (2002)), most of the teams tend to use the option of the initial calibration of their camera system rather than creating more sophisticated adaptive system (Wyeth and Brown (2000)). It is necessary to do an initial camera system calibration in order to get as accurate colour detection as possible, too.. That is typically a manual detection of game field borders and the primary robots and ball colour setting. However, this approach includes a clear disadvantage, as any change in lightning on the game field can lead to a lower quality image acquired by camera and subsequently to a more difficult detection of individual objects on the game field. In such a case, a new calibration has to be done or a mentioned adaptive system has to be used. The adaptive system is much less dependent on light conditions and colours quality, but is much more demanding on the concept itself. The motivation to create a new system of robot marking is the effort to make a system as independent on the light conditions as possible, like adaptive systems, but also preserving the simplicity of the system concept, like currently used camera systems.

LEGO AND LED
The concept presented by us focuses on centralized robot control, therefore including one central camera system. Instead of standard square-shaped robots, this concept uses LEGO MINDSTORMS NXT robots (LegoNXT (2011)). These are, because of their simplicity, used for example in Robocup competition category for children from 7 to 14 years of age (Lund and Pagliarini (2000)). There is a detailed description of our concept, which uses mentioned Lego robots together with colour marking method made of LED diodes, in the following chapters.

Height	160mm
Width	150mm
Lenght	200mm
Weight	800g
Wheels base	87mm
Wheel diameter kola	56mm

Table 1: Robot Parameters

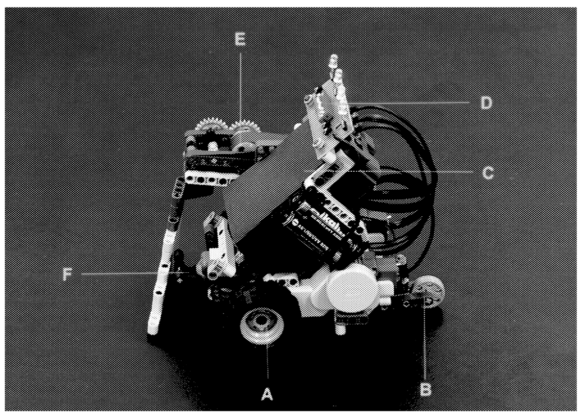


Figure 1: Resulting Robot Design

ROBOT CONSTRUCTION AND ROBOT AND FIELD MARKING

For the construction of the robot football (hockey, floorball) player, we used LEGO MINDSTORMS NXT construction set. The robot look is inspired by a basic tribot model. It is a wheeled robot using two engines (A, Fig. 1), enabling so called differential driving. In order to stay stable, the robot is equipped with a small wheel on the back (B, Fig. 1), which rotates, based on the movement direction. From the third engine, a robot's hand was made by using gears. It is attached on the right side of the robot and it is possible to attach e.g. a hockey stick to it (E, Fig. 1). A touch sensor detects whether the hockey stick returned back to its place or not (F, Fig. 1). The Table 1 includes the final robot construction parameters.
The MLCad tool (http://mlcad.lm-software.com) was used to create a virtual model of a robot player. This tool enables model construction using three projections and can display the model in 3D view. For watching the final model, a tool such as LDWiew (LDView (2011)) can be used then. The way of driving a LEGO NXT robot was realized using Microsoft Robotics Developer Studio (MRDS (2011)). Microsoft includes service

for controlling several typed of robots and construction sets, as well as LEGO NXT, in the basic installation of MRDS. The whole architecture of MRDS is based on services. The basic idea is the change of robot control subject to subject of orchestration of individual services representing the control logic or robot's hardware (sensors, driving units). The communication between the control unit NXT Brick (C, Fig. 1) and a PC is possible only with using Bluetooth in MRDS. The NXT control unit includes a programme providing communication with a PC, on which a control application is hosted. On the PC, the LEGO NXT Brick application provides communication with the robot.
A coordinate system with a start in the middle of the game field was used for calculations. In the calibration part of the testing application it is, considering further calculations used for example for elimination of the shift caused by diodes placed on the top of the robot, necessary to set game field corners.

ROBOT MARKING
The robot's colour marking is used for its identification on the game field. Each robot is labelled with a code consisting of four positions. Each one of them can be coloured differently from the others. A camera then detects robot's codes and positions. The first colour is a reference colour; every robot on the field has the same. This colour guarantees a correct detection of a scanned robot's code. This position is also used as a reference to determine the position of a robot. Therefore, the use of this colour is limited to the first position only. The last colour defines team. The second and the third colour define robot's number in the team. An example: a code is used, using three colours, one of them being the reference one. There are two colours left for team number definition, and two possible locations, which is $2^2 = 4$, which means that a team can have only four members. The player's number is set binaurally; one colour is regarded as logical 1 and the other one as a logical 0. It is also possible to raise the number of positions in the code. That would increase the number of possible combinations. The Table 2 presents the meanings of each code position.

Position	Meaning
1.	reference colour
2.	team player identity
3.	
4.	team color

Table 2: Position Meaning

To identify a maximum of four players in a team on the game field, a code using four positions and three colours, for example red, blue and green, is necessary. If red is chosen as a reference colour, blue is a logical 0 and green logical 1 then the code "red, blue, green, blue" stands

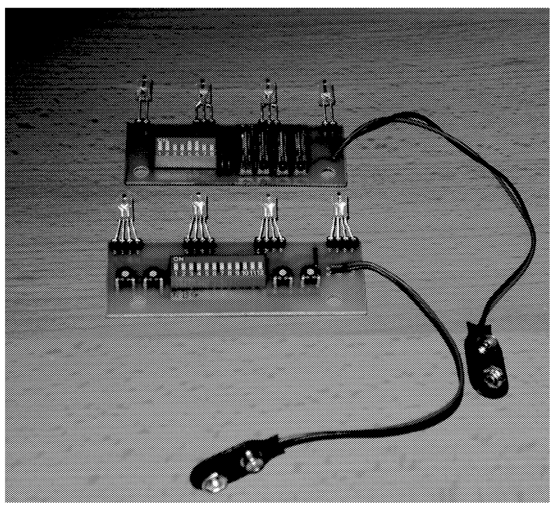


Figure 2: LED Modules

for the first team's player number one. Other possible player markings are summarized in Tab. 3.

Team	Player	Colour code
1.	0	RBBB
1.	1	RBGB
1.	2	RGBB
1.	3	RGGB
2.	0	RBBG
2.	1	RBGG
2.	2	RGBG
2.	3	RGGG

Table 3: Possible Marking with Three Colours (RGB)

The system described above is realized using a module containing of four multicoloured LED diodes (D, Fig. 1). The module is made by printed circuit, which is attached to the top of a robot. The diodes are attached to the module using a precise bar, in which are the diodes inserted. That allows an easy change of defective LED diodes and a possibility to test different LED diode types. The module is powered by two batteries, which provide the module a voltage of approximately 3V, and is set with potentiometers enabling voltage regulation for each LED diode. The regulation of diode voltage decrease causes the LED diode brightness regulation. To determine which colour on a LED diode to light, the module is equipped with a small DIP switch. Two versions of this module were, gradually, created for testing. The first module was used only for testing appropriate diodes. The second module can be equipped with four tricoloured LED diodes. Larger printed circuit and DIP switch were used due to an increased number of LED diodes outputs and potentiometers were changed for classic micro potentiometers (Fig. 2).
By using suitable types of LED diodes, it was discovered that diodes that are the easiest to recognize are

pure or made from milk glass, and have brightness of $1 - 4$ candela. The advantage of those LED diodes is that their colours also respond to colour channels of the RGB model and therefore simplifying the following image analysis.

ROBOT IDENTIFICATION

The algorithm describing robots position on the game field has to solve several problems, using our method (marking robot with LED diodes and placing the cameras on sides of the field (Fig. 3)):

1. Detecting LED diodes in a camera image.
2. Distance distortion elimination using projective transformation.
3. Elimination of shift caused by positioning LED diodes on the top of a robot.
4. Getting a correct rotation and an ID of a robot.

The next part of this article deals with finding a colour code representing a robot in a camera image. During solving this task various colour models (RGB, HSV) and parameters setting options were tested. As emerged from these tests, the HSV model should be used due to a better colour specification in case of bad light conditions. The algorithm for locating colour blobs representing a robot in the image consists of the following steps:

1. Processed image loading.
2. Setting the search parameters.
3. Dividing on colour levels according to HSV model.
4. Creating a mask for Saturation(S) and Value(V).
5. Selection of potentials blobs based on their colour tone.
6. Mask application on potential blobs.
7. Blob filtration based on parameters.

The parameters for search are defined by a group of allowed colours and bordering the search area. This bordering corresponds with the game field area. These parameters are obtained in the calibration part of the application. The camera image is divided into colour levels. That creates three images in grey shades. These are single-channel representations of individual HSV model components. Points belonging to each level then represent component values in the particular image point. The next step is to create a mask consisting of Saturation (S) and Value (V) layers. The mask represents a group of points having required brightness and intensity values. The method for creating the mask assigns value 1 to a point in case its value is higher than the border; otherwise the point is given value 0. The mask can be displayed as an image, which contains white spots where S and V reach the value required. The rest of the image is black.
Localization of points with appropriate value of colour tone H is carried out in a similar way. The method used assigns value 1 to every point in an interval given, while value 0 is given to points outside the interval. The final blob map for a certain colour is made using a logical sum of a mask and a layer acquired in the previous step. In order to obtain colour blobs it is essential to choose blobs, which might represent LED diodes. The criteria are: size, position and shape. Thanks to this, blobs too large, reflections, points outside the game field and points not having the brightness required, will be removed. Several variations of testing were used. After testing various LED diodes and the camera, experiments with colour code identification analysis took place, using static pictures and later video. These tests were conducted in various light conditions and with varied camera positions. In dark, the method is very successful. In case of a high concentration of brightness in the image, the algorithm may evaluate some other parts of a robot as a LED diode representation. Robot consists of, for example, white shiny components such as the control unit, which could be identified as a LED diode. One of the solutions was to cover these parts with paper.
The disadvantage is that the middle of the diode shines with white light and the colour chosen is visible only on the edges. This problem was the worst at blue colour. Another problem is the relatively small size of the diodes compared to the total camera image size. If the camera captures the game field from a too sharp angle, some parts of the robot can appear as diodes. This effect happens the most to red colour, as red is the commonly used colour for LEGO parts. During searching, the algorithm tries to solve the problems described using diode size and shape check. Besides that, a secondary filtration of points found, when points too far away from the others are eliminated, takes place.

CAMERA POSITION AND GAME FIELD SCANNING

As the game field is scanned by the camera sideways and not from above (Fig. 3), the final image is affected by projective transformation (the perspective). The objects farther from the camera then look closer to each other, more than they actually are. That results in inaccurate distance measuring and, therefore, inaccurate setting of the robot's position. That is why it is essential to remove this effect.
For this reason it is, first of all, necessary to find out, in which particular way the image is deformed by the camera, and then describe this transformation, which will convert the image into the form required. That

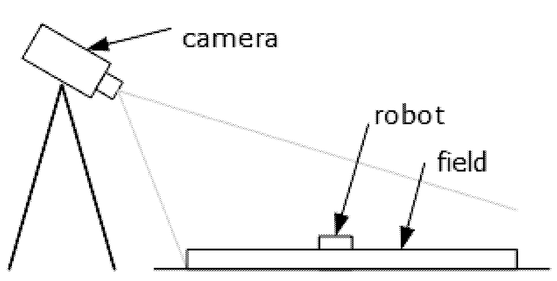


Figure 3: Camera Position

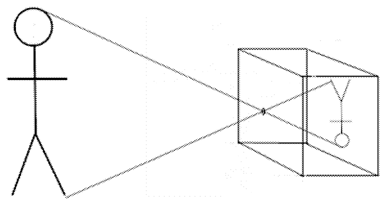


Figure 4: Pinhole Camera Model

is possible by creating a transformation matrix by using the knowledge of the position of four points in the source (deformed) image and their corresponding position in the target (non-deformed) image. In practice that means creating the matrix, which is able to map the coordinates from the deformed image to the real coordinates of a field sized.

CAMERA MODEL

The relation between an object in the real world (3D space) and its image acquired by the camera (2D space) is described by the so called pinhole camera model. In this case the camera is represented by a box, which has a hole in one side. The objects are then projected through this hole on the back side of the box (Fig. 4). In case of the basic model, an ideal camera is used; the size of the hole is modeled as a single point, which all light rays go through.

In the real world, however, the model is more complicated, because it is distorted not only by the camera model, but it is influenced by the imperfections of the lenses as well. This distortion is evident especially on the edges of the image. Despite that, this model gives a good idea of the work of the camera. From the figure 5, it is clear that the image taken by the camera does not provide completely accurate information about the real object. This projection could be described as a projective transformation followed by a 180° rotation of the object. The rotation itself is not important, as the camera output is often turned back of the angle. As for image analysis, it is important to consider the impact of this projection on the image acquired.

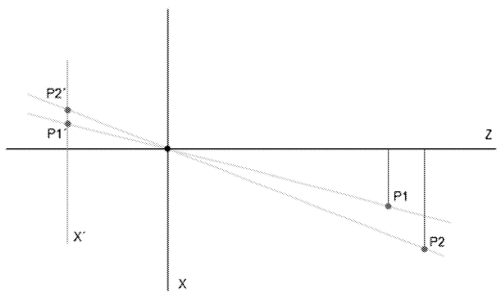


Figure 5: Geometric Interpretation of Pinhole Camera Model

PROJECTIVE TRANSFORMATION

Projective transformation falls into a group of geometrical image transformations (changing the shape or the size of the object). This transformation keeps the linearity of the structures transformed (lines transform into lines), but what it does not keep is their parallel. This effect is well noticeable in the Fig. 5, from which it is evident, that the ratio of the distances between the axis Z and the points $P1$ and $P2$ was not preserved at all. Projective transformation is used especially in cases which, in some way, relate to central projection. An example of a practical task is a transformation, using which the image should be deformed in such a way, that it would seem that it is placed on a generally placed plane in space, which we display by central projection. The second typical use is the opposite of the method described above; it is used for removing the deformation caused by central projection. This deformation removal is exactly what needs to be done in order to identify robots on the field correctly. Since the correct size of the field is known (130×150 cm), the deformed image from the camera will be transformed in such a way that the coordinates from the image are mapped on the particular positions on the field. In order to create a transformation matrix, we have the following options: to run an algorithm for finding a player's identity and then transform the acquired coordinates of the reference point, or, on the contrary, to transform the entire source image into the target image and then run the algorithm for reading the player's identification code reading.

HOMOGENOUS COORDINATES

Homogenous coordinates are used in the description of projective transformations. This coordinate system is used in projective geometry instead of the classical Cartesian coordinate system. Its main advantage is the ability to describe a point placed in the infinity using finite coordinates. That represents a key advantage for projective geometry, as it enables matrix operations, which could not be performed with points in the infinity otherwise. Homogenous coordinates describe a point in an $n-$dimensional affine space as a direction in an affiliated $(n+1)-$dimensional vector space. As an example, a two-dimensional affine space could be thought. The homogenous coordinates of the point (x, y) belonging to the space will be: wx, wy, w. As w, any real number can be chosen, but it must be different from zero. In a homogenous coordinate system, a point is represented by an infinite number of vectors belonging to an affiliated vector space. In practice, it is recommended to choose $w = 1$ as the final triple is $(x, y, 1)$. A conversion from homogenous to affine coordinates can be achieved by the opposite process so that is by dividing individual vector components by the last component.

ELIMINATION OF PROJECTIVE TRANSFORMATION

To eliminate the distortion it is necessary to describe a transformation, which can convert the image into the required (non-distorted) form. It is realized through a matrix, which can be created using the knowledge of four points in the source image and their positions projected into the target image. The relation describing the projective transformations of a point with affine coordinates (x, y) is expressed in the following way:

$$\begin{bmatrix} \varphi \\ \psi \\ \omega \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The relation shows that the homogenous coordinates of the point after the transformation will be $(\varphi(x, y), \psi(x, y), \omega(x, y))$. For their conversion to affine coordinates, it stands

$$x' = \varphi(x, y)/\omega(x, y),$$
$$y' = \psi(x, y)/\omega(x, y).$$

The projective transformation itself is described by nine values $p_{11}, ..p_{33}$. To acquire them, the knowledge of two fours of points mentioned above is needed. Points in the input image can be named for example A'_1, A'_2, A'_3, A'_4 and the points in the output image could be named A_1, A_2, A_3, A_4. In addition, it is essential to establish the marking $P_1 = (p_{11}, p_{12}, p_{13})^T, P_2 = (p_{21}, p_{22}, p_{23})^T, P_3 = (p_{31}, p_{32}, p_{33})^T, X_i = (x_i, y_i, 1)^T$. Based on the relation defining projective transformation, we get

$$\begin{bmatrix} w'_i x'_i \\ w'_i y'_i \\ w'_i \end{bmatrix} = \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix} X_i$$

It is possible to derive $w'_i = P_3^T Xi$ from the third equation of the relation. After appointing to the first two equations and an adjustment, we get

$$\begin{bmatrix} X_i^T & 0 & -x'_i X_i^T \\ 0 & X_i^T & -y'_i X_i^T \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

As each one of the points contributes to the finding of the transformation matrix with two equations, eight equations with nine unknowns are created. And since the transformation matrix can be multiplied by any number except zero without the change of the result, the value of one of the elements can be set as any value (for example 1). The difficulty of this solution is a situation, when the real value of this element equals zero. In that case the equation system would be insoluble.

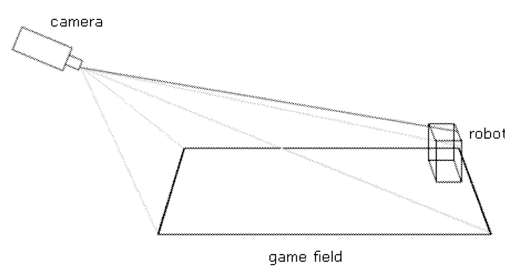


Figure 6: Interaction Between Robot Height and Camera Position

THE EFFECT OF ROBOT'S HEIGHT ON THE CALCULATION OF ITS POSITION

Since the camera scanning the field can be placed on one side of the field and the position is calculated from positions of LED diodes found in the image, the physical location of these diodes on the robot has to be taken into consideration. In our case the module attending to LED diodes is placed on the top of the robot (Fig. 1).

From the definition of the pinhole camera model, it is evident that the height of a robot has a fundamental impact on LED diodes projection in the image captured. This situation is shown in the Fig. 6. In the image captured by the camera, the position of diodes seems to be out of the game field, even though the robot is located on its edge. If this problem stayed unsolved, the algorithm would not be able to find the accurate position of the robot. The accuracy of finding the position would then depend on the physical location of the camera. If the camera captured the image from above, the coordinates acquired would be relatively accurate, though the accuracy would decrease with bigger distance of the robot from the camera. With camera placed as in the Fig. 6, all of the positions would be set incorrectly.

In case of neglecting the effect described above, it would be assumed that the LED diodes are located on a plane, which is formed of the field. The solution to this problem is to find a plane, which is parallel to the field, but also shifted by the axis Z upwards by the value equal to the height of the robot itself. This plane is shown in the Fig. 7.

SOLUTION OPTIONS

One of the solutions proposed is to place lighthouses (or other well-recognizable marking) on the edges of the field in height which is equal to the height of the robot.

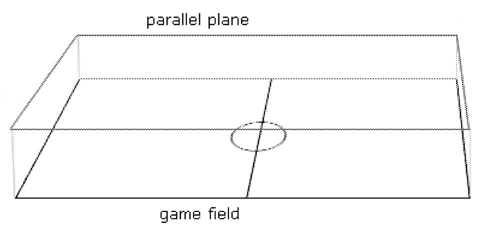


Figure 7: Plane Containing LEDs

An advantage of such solution is its relative simplicity, as it is sufficient to get the position of the lighthouses in the image (using the algorithm for finding blobs) and then to continue with the elimination of perspective projection. The disadvantage is, however, the necessity to adjust the field every time, when the robot's height is changed. The second solution is to mark the corners of the plane in the calibration part of the application. The problem of this method is the difficult determination of accurate positions of these points. Besides, it is a manual process which requires a correct estimation of the distances in the image and may, therefore, unnecessarily bring a mistake to the transformation. Our solution consists in marking the field corners manually in the calibration application. The plane containing LED diodes is then automatically calculated. The manual corner marking is much easier than estimating, how high should be the diodes located. Moreover, no further game field modification is needed.

IMPLEMENTED PROCESS OF FINDING THE PLANE

The process implemented to our system could be summarized in the following steps:

1. Detection of physical proportions of the field in the real world, including its length, width, position, goal size etc.
2. Detection of the robot's height and therefore the value of shift in the axis Z of a virtual plane with LED diodes.
3. Acquiring coordinates of the corners of the game field in the image using mouse click in the calibration part of the application.
4. Camera matrix calculation.
5. Calculation of coordinates of points bordering the virtual plane with LED diodes.

The first step is acquiring the coordinates, which represent all the necessary points in the real world. In order to acquire the coordinates of the virtual reality located above the field, it is only necessary to add the robot's height to the Z coordinate of individual vertices. The coordinate of game field corners is acquired using the calibration part of the application. It is a manual part, in which the user marks the corners in the camera image by mouse clicking. The following matrix calculation is described above. Using this matrix, it is possible to determine where will be the points bounding the plane with LED diodes. 3D coordinates of individual field corners and their 2D representation acquired from the user in the calibration part serve as inputs. The last step is multiplying the matrix with individual points located above the corners of the game field. The result of this multiplying is a representation of points in the image set using homogenous coordinates. The coordinates acquired this way are then used in the algorithm for finding blobs in the image, where they serve as boundaries of the area, in which should be the blobs searched for. Among other things, they are used in the algorithm for eliminating perspective projection.

DETERMINATION OF ROBOT'S CODE, POSITION AND TURNING

After processing the image by individual algorithms, the application gets a group of light points of various colours and their position in relation to the field. The last step which needs to be done is therefore reading the robot code, thereby acquiring their positions on the field together with their turning direction. The target of this algorithm is creating appropriate $n-$tuples, where n is the number of diodes used to identify the robot. After creating this $n-$tuple it is possible, thanks to the knowledge of the meaning of individual diodes, to derive its explicit identifier and turning. The group acquired may also include points, which do not represent diodes. These can be for example mistakes of the algorithm for colourful blob finding in case of bad light conditions. For this reason it is necessary to check also the distance between points found and the mutual angle, which the lines going through the pair of points and the X axis hold. In case the diodes belong to a single robot, the angles should not be very different. The algorithm starts with a choice of one of the reference points (reference diodes) and then looks for the nearest non-reference point, which meets the distance requirement. After locating such points it continues to look for the next non-reference point until the whole $n-$tuple is found. It is also checked if the points found lie approximately in a line. In case it is not managed to decode that $n-$tuple successfully, looking for the next reference point follows.
As the position of the robot itself, the centre of the line segment found is chosen (Fig. 8). The orientation is presented as the angle between a line (going through the first ant the $n-$th point found) and the axis X, from which $90°$ is subtracted.
Two robots on the game field are shown on Fig. 9. The game field is not large enough for more then two

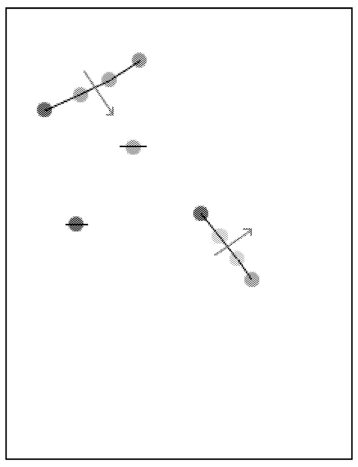


Figure 8: Resulting Robot Position and Rotation

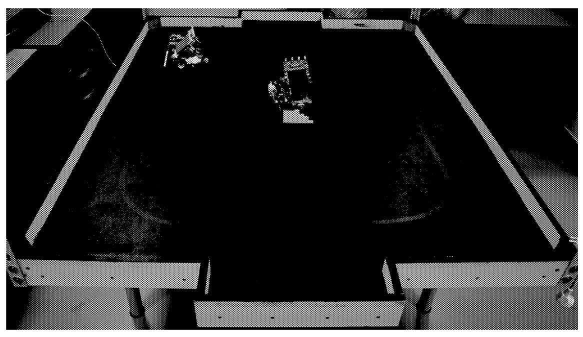


Figure 9: Two Robots

players in each team. However, presented method does not depend on the game field size or robots number but on the quality of image analysis.

CONCLUSION

This article describes a new possible way of marking robots in a robot soccer game. Our presented method uses colored LED diodes for easier identification of robots on the field and also includes a dynamic camera system for image recognition. Because the camera is positioned on the side of the field instead of commonly used central position above the field there is also introduced the method used to remove distortion caused by projective transformations. By replacing traditional paper-based labeling for light-emmiting diodes we can achieve significantly better results in the detection of objects during image analysis. Thanks to this light their identification in the acquired image is relatively easy even though the game takes place in a room with not too good lighting conditions. Their use brings completely new possibilities for image analysis as well as for the actual robot soccer game. For example if we use LED diodes to mark both the robots and the field and also found a way how to create a glowing ball it is not hard to imagine a game taking place in total darkness. For future work we want to use diodes not only for Lego robots but also in Robocup SSL category so we could create whole game based on this marking system using LED diodes.

REFERENCES

Anders J. Johannson K.R.G. and Balkenius C., 2002. *Lighting of RoboCup Games. LUCS.*

Gerd Mayer H.U. and Kraetzschmar G.K., 2004. *Playing Robot Soccer under Natural Light: A Case Study. 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence.*

Gregor Klanar Omar Orqueda D.M. and Karba R., 2001. *Robust and efficient vision system for mobile robots control - application to soccer robots. Elektrotehniki vestnik 68(5).*

http://code.google.com/p/ssl vision, 2011. *SSL-Vision Developer Team: RoboCup Small Size League Shared Vision System. Project Web Site.*

http://small-size.informatik.uni bremen.de, 2011. *RoboCup Small Size League: SSL Web Site.*

http://www.fira.net, 2011. *Federation of International Robot-soccer Association: FIRA Web Site.*

LDView, 2011. *LDView.* URL `http://ldview.sourceforge.net`. (2011.07.13).

LegoNXT, 2011. *LegoNXT.* URL `http://mindstorms.lego.com`. (2011.07.13).

Lund H.H. and Pagliarini L., 2000. *RoboCup Jr. with Lego Mindstorms. IEEE International Conference on Robotics and Automation.*

MRDS, 2011. *MRDS.* URL `http://msdn.microsoft.com/en-us/library/bb881626.aspx`. (2011.07.13).

Sridharan M. and Stone P., 2005. *Real-Time Vision on a Mobile Robot Platform. IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Vctor M. Gmez Jos M. Canas F.S.M. and Matelln V., 2003. *Vision-based schemas for an autonomous robotic soccer player. IV Workshop de Agentes Fsicos WAF-2003.*

Wyeth G. and Brown B., 2000. *Robust Adaptive Vision for Robot Soccer. IEEE International Conference on Robotics and Automation.*

Zickler S.; Laue T.; Birbach O.; Wongphati M.; and Veloso M., 2009. *SSL-Vision: The Shared Vision System for the RoboCup Small Size League. RoboCup 2009: Robot Soccer World Cup XIII.*

SERIOUS GAMING

DIDACTIC GAMES SYSTEM: FIFTEEN YEARS OF DEVELOPMENT IN MILITARY SIMULATION

Gustavo Henrique Soares de Oliveira Lyrio [1]
Roberto de Beauclair Seixas [1,2]

[1]PUC-Rio–Pontifical Catholic University
TECGRAF–Computer Graphics Technology Group
Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brazil

[2]IMPA–National Institute for Pure and Applied Mathematics
Estrada Dona Castorina 110, 22460-320 Rio de Janeiro, RJ, Brazil

emails: glyrio@tecgraf.puc-rio.br
rbs@impa.br

KEYWORDS
Warfare Games, Military Modeling and Simulation,

ABSTRACT

This work has the goal of presenting the author's experience through the last fifteen years developing military simulation games for the Brazilian Marine Corps. Starting from the birth of the idea of a didactic training game, we pass through the problems found and the solutions adopted to nowadays, where the system covers all kinds of attributions that a Brazilian Marine officer could receive.

INTRODUCTION

The history of war games are as old as the history of organized conflicts. he first use of a war game as a teaching tool is said to be a game developed by the Prussian Guard artillery lieutenant George Heinrich Rudolf Von Reisswitz and his father, Reisswitz Baron, to teach Prussian Guard officers. (Wilbur 1995)

In Brazil, the first notice of wargamming came from a small volume stored in Escola de Guerra Naval ("Naval War School") library called "Como jogar o Jogo de Guerra Naval" ("How to play the Naval War Game") dated from 1915. Since that year, war gaming was a reality to Brazilian Navy officers, first as sand boxes, board games, and then maps and computers after 1985.

The Brazilian Marine Corps, which is a division of the Brazilian Navy had foreseen the necessity of their own didactic war game system in 1997. Then, a partnership program was started with Tecgraf/PUC-Rio to its development.

The next sessions will present the journey through this years of civilian and military joint work, making a overview of the system developed, the obstacles found and the solutions adopted to make this partnership a case of success through all this years.

THE CHALLENGE OF THE FIRST YEARS

It's common sense that the first step is always the hard one. That was truth in the case of the development of a Didactic Game System that could achieve the goals of the Brazilian Marine Corps. As said above, in 1997, other systems that already use computer simulation were running in the Brazilian Navy, but nothing like the Marine Corps had expected. The system running at the Naval War School was conceived (and still is) to teach sea operations, which means flat surfaces, low detailed maps and lots of simplifications.

Those simplifications could not be applied when taking trainment of Marine officers in consideration. The modeling should start by the most common operation, which is also the most complex of all military maneuvers: Amphibious Operations. This kind of warfare means the use of naval firepower, logistics and strategy to project military power ashore and allow the landing of troops in a non-contiguous enemy-held terrain.
The idea was that the new system should be able to allow the training of Marine officers in Amphibious Operations with low cost, avoiding the necessity of sending troops to real terrain. With that in mind, the system's specifications became our first problem. The request for development of the Marines' new system came with a lot of complex, yet relevant, demands like 3D real terrain, high detailed land with different kinds

of soil, vegetation, climatic and astronomic conditions and everything that could affect a ground war theater.

In technical terms, the system demands are mainly a high detailed GIS (Geographic Information System) software with large real 3D terrain models (currently 21 layers of information). Everything should be persistent and consistent with the way Brazilian Marines conduct their operations making the system able to be compared to real training. All that just in 1997!

When the first contact between the Brazilian Marine Corps and Tecgraf/PUC-Rio was made, the idea was evaluate if there were technology available in Brazil to do something as huge as such system demanded. Then, a PhD was requested to know the system specification and to write a report either indicating which research institution could perform such task or suggesting the acquisition an existing foreigner system.

After 3 months, the first version of what would became the Didactic Games System was born. The Marines became satisfied with what they saw and decide to sign the first partnership contract and proceed with the development.

In this version (Figure 1) we presented some units positioned, using UTM (Universal Transverse Mercator) coordinates, in a real terrain that were able to move, see each other and interact thought mathematical models based on Lanchester equations to determine casualties. This system of differential equations is based on the principle of losses imposed to one force are proportional to the number of elements in the other side.

$$dx = \frac{-aY}{dt}$$

$$dy = \frac{-bY}{dt}$$

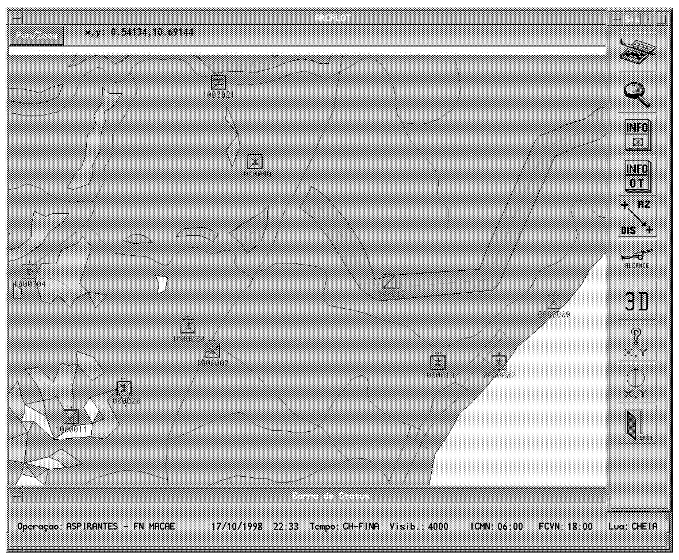
Lanchester differential equations Where a and b are the efficacy coefficients of the forces Y and X, respectively.

$$ACP = \sum_{i=1}^{all\ weapons} (qty(i)cad(i)let(i))$$

Combat power equation where qty is the amount of weapons, cad is the cadence of the weapon and let is the lethality of the weapon when using specific ammunition.

The main drawback in using Lanchesters model is the fact that the correct values of a and b depend on: weapons characteristics, elements' capacity of reaction and on command and control. We choose to made our engagement model determining how the accumulated combat power (ACP) of each element will be calculated which lead us to the previous equation.

The system was composed by four different components: database, geoprocessing, engine and user interface. After a few years using commercial softwares, was necessary to reduce budget to spend more with hardware update. The natural decision was to migrate to free software. So we chose Linux as the operational system.



```
Operaçao: ASPIRANTES - FN MACAE   17/10/1998  22:33  Tempo: CH-FINA  Visib.: 4000   ICMN: 06:00   FCVN: 18:00   Lua: CHEIA
```

Figure 1: Screenshot displaying the first version of system interface.

For the database, the first selection was Informix because at that time it was also free. The user interface received a special treatment due to the fact the officers that would use the system were not familiar with Linux. So we agreed to make it for Windows but with the restriction that it had to be portable further.

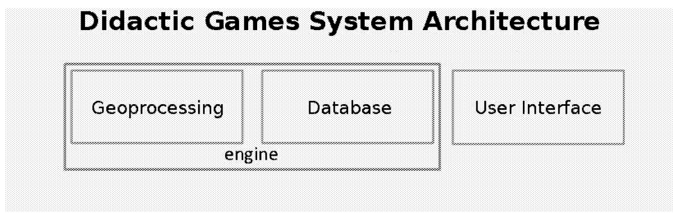


Figure 2: Schema illustrating the first system architecture.

The work on the following years was something unexpected for both parts. The Marines start to teach us how to be a Marine and, in return, we start to teach then how to be software developers.

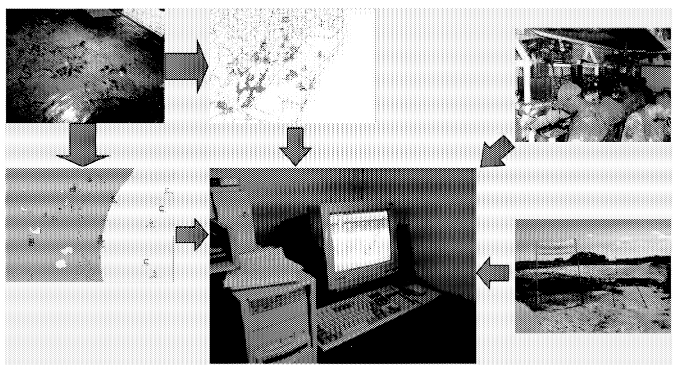


Figure 3: The goal of the project was to make everything available to the officers inside the simulation.

SUCCESS BRINGS MORE WORK AND EXTRA RESPONSIBILITY

In 2002, with the system fully operative, the goal was achieved. The Didactic Games System became reality and it has been used even by the Brazilian Army in joint exercises.

Later came the idea of extend it to cover other than amphibious operations. The first change requested was that the system should cover both Amphibious and Riverine Operations. The choice for Riverine operations prior to other Marines' attributions was due to the fact that a large part of Brazilian territory borders is covered by the Amazon Basin, the world's largest river basin and also the world's largest rain forest and Pantanal(both riverine terrain), making it a constant hideout for drug dialers and paramilitary forces. Also the local population suffers from a lot of necessities (like basic health service) due to difficult access to this area by the Brazilian Government. All that characteristics made the Riverine Operations the second most common operation for Brazilian Marines, and our second modeling goal.

Adapting the system to Riverine Operations was an entirely new challenge. This because while the first system had no predecessors in Brazil, in the case of riverine operation simulation systems the authors didn't know about the public existence of such a system anywhere in the World.

The responsibility increased because the partnership has proved that if it was possible to build a system to cover Amphibious Operations and to publish papers about the technology been used, it would be also possible for us to extend it to cover Riverine Operations.

Research results achieved by the project until 2002 were 1 Phd thesis, 4 master dissertations and 8 published papers.

The first and most radical change that we have made in this phase of development was to change the engine development language from INFORMIX-4GL to Lua. This decision was taken due to the necessity of the system to be portable, and INFORMIX-4GL had been a language that came from the choice of a particular database system. The choice for Lua was very natural because we were already using it to build user interfaces. In addition, Lua's fast curve of learning and the fact that it was developed inside Tecgraf/PUC-Rio helped us in the choice of the system development language. So, we gained speed in new developers training and in technology transference to the Marines. Thus, we kept the entire system wrote in a single language, reducing the specialization requirements for new developers in order to be involved in the project.

That was a blessed decision because few months later Informix was brought by IBM and we decided to change the Didactic Games System database to PostgreSQL. Following, PostgreSQL started to be part on all Linux distributions providing kernel integration, which turned the communications between the interface and engine even faster. Also we detached the geoprocessing component from the engine and moved it to pre-processing phase, with that we made the engine, the database and the user interface independent from each other.

Another change required was relative to the terrain. In Riverine Operations terrain changes completely when compared with Amphibious Operations. The terrain may change even when comparing different riverine terrains, like Amazon (mostly composed by dense jungle and rivers) and Pantanal (a huge plain of flooding areas), for instance. Besides the terrain characteristics, it was necessary to improve the terrain details due to the fact that in Riverine Operations the troops are divided into small fractions. Add to that the difficulty in obtaining detailed digital data of this kind of region, such as satellite images.

The solution came in applying a quantization algorithm to the satellite images to reduce the colors to the number of different types of soil and vegetation needed in the simulation.
Additional adaptations were made to turn the previous version of the system capable of simulating a riverine operation. It worth to mention here the modeling of the local population. Its role is extremely important in this kind of operation due to the fact that the guerrilla soldiers usually are recruited in the local people, making then look like, dress like and speak like local people. The only difference between a local citizen and a guerrilla soldier in this kind of area is the fact that the soldier is carrying a weapon. The guerrilla knows and makes use of it. They hide their weapons in the jungle,

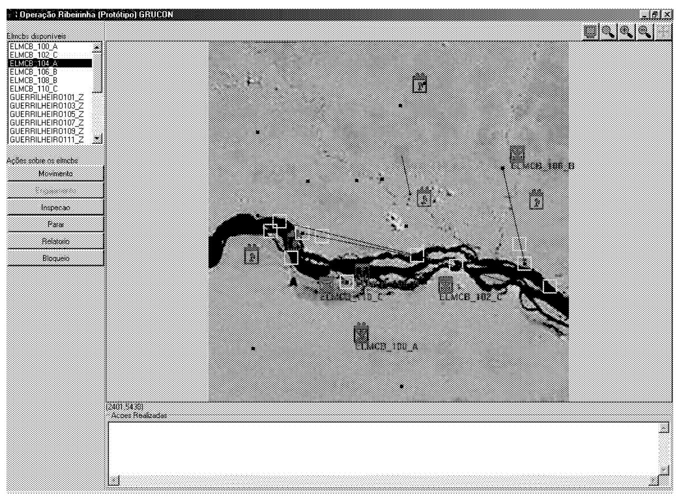


Figure 4: The Riverine Operations interface showing enemy, friend and neutral elements spread over the Amazon.

in the boats, and try to look like innocent persons walking side by side with the population. That fact made the local population a must have characteristic in riverine operations simulation.(Figure 4).

With the necessity of modeling the local population, came the demand of some artificial intelligence to take control of this new role (neutral). Until now the system had two well defined sides (friend and enemy) each one controlled by users. To accomplish that, we developed an artificial intelligence library based on agents with predefined behavior called MARE (Modeling Agents for Real Environments) (Lyrio and Seixas 2007), whose objective is not only to take care of the local population in Riverine Operations, but also to deal with the fact that the Marines, at least in Brazil, are not prepared to defend (Figure 5). In other words, the officers who took the role of the enemy during the simulations wasted part of their training time with a task that they usually were not trained for.

It is important to notice that we had to develop a new engagement model over the Lanchester equations in order to adapt it to guerrilla and also to respect the relationship between friend and enemy combat power of 10 to 1, adopted when taking non-conventional enemy forces in consideration (to be consistent with doctrinal rules). To achieve that we added a factor representing the number soldiers in the friend troop to the first Lanchester equation that would represent the guerrilla and leaved the second one unchanged to the conventional force.

In 2004 we published a paper and delivered to the Marines the first version of the Didactic Games System supporting Riverine Operations. (Lyrio and Seixas 2004)

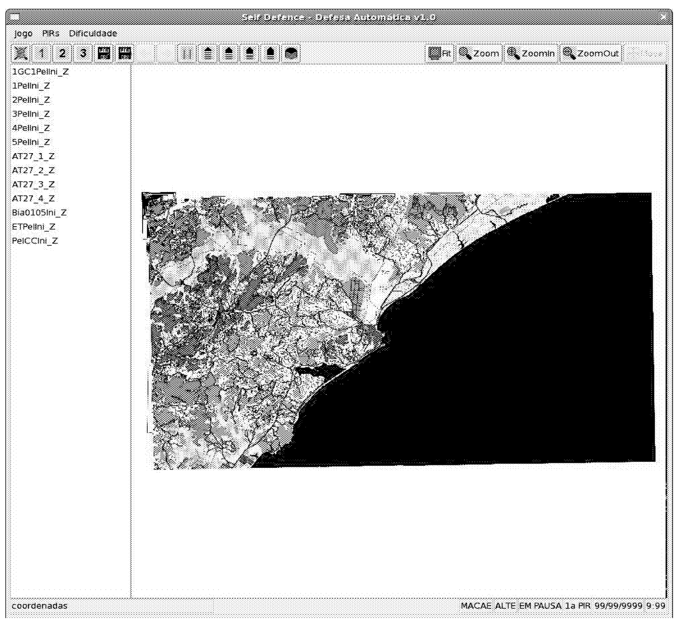


Figure 5: Automatic defense interface using Brazilian Navy graduations to define difficulty levels.

COVERING ALL THE DUTIES - INCURSION AND OPERATIONS OTHER THAN WAR

After the success achieved with the Riverine Operations simulator, we receive the request do extend the system even further covering all the duties a marine officer could receive. The goal was then to model Incursion and Operations Other Than War (OOTW), which means urban terrain operations.

In few words, Incursion is an operation where the marines have to enter a hostile territory, do some fast actions like hostages rescue or target elimination, and leave. We can take as example the action in the Japanese embassy hostage crisis in Lima, Peru.

By operations other than war we understand the actions to keep or recover peace in a territory, in special or critical moments. Again, as an example we can take the actions recently made by the United Nations (UN) in Haiti.

We put this two apparently different kinds of operation in a single session due to the fact that their modeling and the difficulties that we found were common in both cases. The most important difficulty that appeared during the process of adapting the system was to deal with constructions.

In the previous operations a construction was not relevant because the actions were all taken in countryside. Now the constructions play a main role in the scenario. For instance, lets take our first example and imagine

$$dx = \frac{-aXY}{dt}$$

$$dy = \frac{-bX}{dt}$$

$$\frac{dy}{dx} = \frac{b}{aY}$$

$$\int \frac{dy}{dx}\, dt = \int \frac{b}{aY}\, dt$$

$$aY^2(t) = 2bx(t) + M$$

or yet

$$M = ay_0^2 - 2bx_0$$

If $M < 0$ then the guerrilla wins. If $M > 0$ the marines win. If $M = 0$ there is a tie between the opposite forces. For usual values of a and b, one can conclude that:

$$\frac{y}{x} \approx \sqrt{\frac{2b}{ax}}$$

or

$$y \approx 10x$$

Modified Lanchester differential equations for guerrilla and conventional forces respectively. Note the new factor X on the first equation.

that an embassy was taken by hostiles and the marines need to rescue the people that were working there during the attack. The whole action will occur in the embassy neighborhood. The entire area is now extremely important to the simulation as the officers need to decide how to block streets, where to take position in nearby buildings, how to inspect citizens that pass through the area and so on (Figure 6). There is a movie called Black Hawk Down that shows how complex may be an Incursion.
In Operations Other Then War, the construction play similar role. The goal in this kind of operation is to help citizens oppressed by a local force, a guerrilla or after a disaster.

Constructions became a problem because, as far as we know, there is no available data showing the buildings that compound the Brazilian marines training areas. To overtake this difficult we made, by hand, a model of each block of the urban training areas estimating an average height based on the amount of houses, buildings and other constructions on the block (Figure 7). The results, despite the fact that are far from reality, was accepted by the Brazilian Marines as a temporary solution, making possible to the instructors to check if the officers were taking correct actions during the operation training.

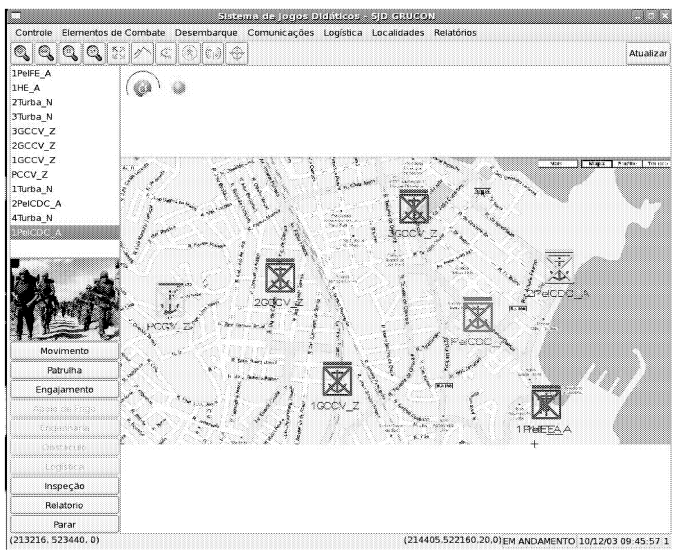


Figure 6: Incursion and OOTW interface modification using Google Maps image to display city blocks.

With the demand for constructions other obstacles arose, as terrain details, for instance. If we had trouble with terrain details in Riverine Operations, the problem now became even worst. In Riverine Operations the terrain was open and mobility was only affected by vegetation permissiveness, so a unit moving along the jungle would have it's velocity changing depending on the vegetation and soil type. Now, with the introduction of constructions, the lack of details about streets and alleys makes the movement during game almost impossible. It's true that a lot of tools like Bing and Google Maps have made incredible street details of almost everywhere in the world. The problem is that such tools do not provide other information like elevation, soil, vegetation, hydrography and other georeferenced layers that the system need to work properly.

Other change that deserves to be mentioned here was the necessity to model a new role in the simulation. Until the development of OOTW the enemy was well defined and lethal force was always used. Now, in most cases where an OOTW simulation is requested we have a role that is played not by enemy, but by unpleased or oppressed groups of citizen that protest or try to satisfy their basic necessities that the government isn't providing. The first approach was to use the local citizens already modeled in Riverine Operations. Despite the fact that in OOTW citizens have an entire new behavior, that wasn't an lost effort.

Most of the time, the riverine citizens are not taking part in the conflict. But, in a particular case they do. Guerrilla some times coerce local people and that is very close to what happens in OOTW. The difference is that when a riverine person is coerced by the guerrilla they became part of it and the citizens in OOTW don't. In OOTW mostly of times people goals are (requested by enemy forces, or to protest) to disturb the order and make chaos in the neighborhood but they are mostly not armed and don't intent to engage the troops.

That brought to us the next improvement in the Didactic Games System. The troops needed to restrain the disorder caused by civilians, something we never thought about during all this developing years due to the well defined enemy behavior in the simulation: the use of non-lethal weapons.

After some discussion we agreed that the best way to insert non-lethal weapons in the simulation was to distinguish it from the lethal ones by the way it affects a target. A surprising conclusion was that it's not so different. The explanation is that when a enemy troop is attacked by lethal weapons it receives casualties. However, when a mob receives a non-lethal attack it also lose men, the difference is that instead of die, they run away and may reorganize in other place after a while. So, based in that conclusion we modeled the non-lethal engagement using exactly the same engagement model we use for lethal except for the fact that the term lethality, in this case, is named power of dispersion. Once a mob lose a percentage of its members (meaning they run away, not died), after some time, we "transfer" this mob to a new position in the terrain near the enemy headquarters in order to simulate a reorganization. With that idea we could insert non-lethal weapons and mobs in the simulation with almost no cost.

$$ACP = \sum_{i=1}^{all\ weapons} (qty(i)cad(i)disp(i))$$

Combat power equation used in non-lethal engagements. Note that the equations is the same as the one presented before expect for the fact that lethality was replaced by dispersion (*disp*) to represent the power of dispersion of the weapon when using non-lethal ammunition.

TURNING THE PROJECT TO THE WEB

During 2008 we have foreseen the possibility of using web 2.0 tools to help to overcome some common problems found during all this years of development. The first problem was how to deal with 3D terrain models of real world areas. We realize that this kind of model would not only demand a detailed model of a real area of the earth but also models of the structures and units that act over this area.

Fortunately, in the same year, Google released the

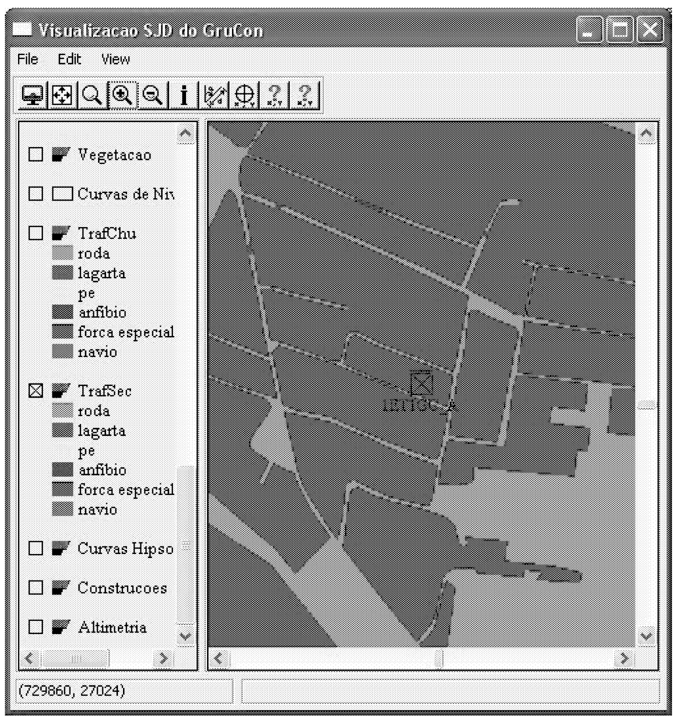


Figure 7: Trafficability map displaying the adopted solution of treating blocks instead of each construction.

Google earth API, which allow developers to display data from Google Earth inside web page. So we choose to use this API and got promising results.

Another issue was how to model troops communication during the simulation with all its characteristics and difficulties like statics, interferences and electronic warfare. Again a solution was found in a web 2.0 tool. In this case we choose to work with Team Speak SDK that became popular in Brazil due to the Counter Strike game, so it was a natural first choice. We also considered that Skype could do the job.

The last problem we faced was related to an user's complain about the lack of ways of keeping track of their units actions during the simulation. The goal of keeping track of units is to permit debriefs where users should be able to discuss wrong line of actions, propose improvements or even show a great maneuver that all other users should take notice. Our line of action in this case was to create a micro blog account for each unit where the units by itself post which actions were taken in the simulation, producing a real time tracker of all the simulation unit by unit. The tool chosen here was Twitter.

A paper called REAL-TIME WARFARE SIMULATION GOES WEB 2.0 was published in 2009 detailing the whole experience. (Lyrio and Seixas 2009)

CONCLUSION

After fifteen years of development we were able to extend the main idea of a simulation system to help training marine officers in Amphibious Operations into a system covering all the Brazilian Marine Corps assignments. The system is capable of simulating climatic and astronomical conditions, movement, communications, fire support, aerial and anti-aerial support, logistics, engineering, non-lethal weapons, electronic warfare, anti-aerial, citizens, mobs, guerrilla, enticement, coercion, endemic diseases and works with the following additional systems and libraries:

- Artificial Intelligence library M.A.R.E.
- Automatic Defense System
- Decision Making Tool to Military Planning
- Maneuver and Attrition Warfare Simulation System
- Command and Control Remotely Monitor System
- 3D Aerial Reconnaissance System

The first conclusion that we take from all this years of development is that the correct choice of the architecture is fundamental. As we mentioned before, we had to replace each of the three parts that compound the Didactic Games System during along these years in different times, and the choice of having this parts independent from each other has made it possible with just a few extra work.

We also learned that having a portable system is important. As the news of the system success reached different locations inside the Brazilian Navy, we received requests to move the entire system to different locations with different hardware and software architectures.

Finally, and perhaps the most important lesson learned is to know well your partners. Because they probably don't know about your development capability, they will not be able to provide all necessary specifications. It's important to by familiarized with partner's demands and necessities in order to build a long and fruitful partnership.

REFERENCES

Campos D.V.; Seixas R., 2011. *Command and Control: A low cost framework to remotely monitor military training.* In *Proceedings of Spring Simulation Multiconference - SpringSim'11, in Military Modeling and Simulation Symposium - MMS.*

Lyrio G. and Seixas R., 2004. *Riverine Operations: Modeling and Simulation.* In *Proceedings of Advanced Simulation Technologies Conference - ASTC, in Military, Government, and Aerospace Simulation Symposium.* 33–39.

Lyrio G. and Seixas R., 2007. *Modeling Agents for Real Environment.* In *Proceedings of The North American Simulation and AI in Games Conference.* EUROSIS.

Lyrio G. and Seixas R., 2009. *Real-Time Warfare Simulation Goes Web 2.0, , , Georgia, USA, 2009.* In *Proceedings of The North American Simulation and AI in Games Conference - GAMEON-NA.* EUROSIS.

Savelli R.M.; Lyrio G.S.R., 2004. *The Maneuver and Attrition Warfare Simulation System.* In *Simpsio de Pesquisa Operacional e Logstica da Marinha - SPOLM 2004.*

Seixas R.B.; Lauro A., 2003. *A Decision-Making Tool to Military Planning Process Based in Dynamic-Cost Matrices.* In *Proceedings of Advanced Simulation Technologies Conference - ASTC, in Military, Government, and Aerospace Simulation Symposium.* 70–75.

Seixas R.B. Mediano M. and Gattass M., 1999. *Efficient Line-of-Sight Algorithms for Real Terrain Data.* In *III Simpsio de Pesquisa Operacional e IV Simpsio de Logstica da Marinha - SPOLM'99.*

Wilbur G., 1995. *Playing War: the Applicability of Commercial Conflict Simulations to Military Intelligence Training and Education.* Master's thesis, DIA Joint Military Intelligence College, Bolling AFB, DC.

BIOGRAPHY

Roberto de Beauclair Seixas works with Research and Development at Institute of Pure and Applied Mathematics – IMPA. He got his Ph.D. degree in Computer Science at Pontifical Catholic University of Rio de Janeiro – (PUC-Rio), where he works with the Computer Graphics Technology Group - TeCGraf. Since 1999 at IMPA, he works as IT Manager and project leader of technical-scientific multi institutional projects. He also is advisor of Warfare Games Center of the Brazilian Navy Marines Corps.

Gustavo Henrique Soares de Oliveria Lyrio works with the Computer Graphics Technology Group - TecGraf. He got his B.Sc. in Computer Engineering at Pontifical Catholic University of Rio de Janeiro - PUC-Rio. His interests include Computer Graphics and Warfare Training Games. Currently he is developer of Warfare Games Center of the Brazilian Navy Marines Corps.

EVALUATING GAMES ENGINES FOR INCORPORATION IN MILITARY SIMULATION AND TRAINING

Jennifer L. Winner
Lumir Research Institute, Inc.
195 Bluff Ave
Grayslake, IL 60030
jennifer.winner.ctr@wpafb.af.mil

Stephen F. Nelson
Ball Aerospace
2875 Presidential Drive
Fairborn, OH 45324
sfnelson@ball.com

2Lt Rebecca L. Burditt
Capt Adam J. Pohl
Air Force Research Laboratory
Wright Patterson AFB, OH 45433
Rebecca.burditt@wpafb.af.mil
Adam.pohl@holloman.af.mil

KEYWORDS
Game engines, modeling and simulation, instructional theory, training effectiveness, distributed mission training

ABSTRACT

A game engine is a tool through which interactive, real-time simulations can be created. Our interest in game engines is motivated by the instructional needs of the United States Air Force (USAF). In this paper we discuss our evaluation criteria for incorporating game engines into pre-existing Air Force distributed mission-training systems. Data availability, 3D model correlation, and ground database integration are primary considerations. If a game engine does not accommodate these criteria as outlined above, it will most likely not be considered for incorporation into Air Force training systems as it would limit our ability to simulate a real world environment. The remaining criteria add significant value to the training environment and help to distinguish game engines from one another.

INTRODUCTION

The level of interest in using gaming technology for training purposes has increased substantially in recent decades, as evidenced by the volume of literature on the topic. Recent studies have shown that experienced video game players exhibit heightened performance on visual perception tasks (Castel, Pratt, & Drummond, 2005; Green & Bavelier, 2007; Green & Bavelier, 2006; Green & Bavelier, 2003; Li, Polat, Makous, & Bavelier, 2009) and, in simulated flight tasks, performance which nears that achieved by pilots on stick and rudder control and target identification and tracking tasks (McKinley, McIntire, & Funke, 2011). Gaming technology has shown to be an effective means for training purposes, including military-focused training (Chatham, 2011) and general teamwork skills (Hussain, et al., 2007).

The documented effectiveness of game-based training is of particular interest to the Air Force Research Laboratory. For the 711th Human Performance Wing, Human Effectiveness Directorate, Warfighter Readiness Research Division (711 HPW/RHA), research conducted in the Gaming Research Integration for Learning Laboratory (GRILL) focuses on the potential use of commercial off-the-shelf (COTS) game and simulation engines for mission qualifying and continuation training for the USAF. Game engines have potential to serve as either as stand-alone training systems or as complementary extensions of existing high fidelity simulation and training environments (Pohl & Walker, 2010). For example, one could imagine a stand-alone system in which an individual pilot may run through checklists and gain practice relating to specific button functionality. Additionally, gaming technology could be utilized to supplement the capabilities offered by high fidelity simulator systems such as the Mobile Modular Display for Advanced Research and Training (M2DART). The M2DART provides high-fidelity simulation of a four-ship of F-16 jets. Instructors wanting to simulate threat tactics could utilize gaming technology and/or low-cost COTS simulation technologies such as X-Plane® to effectively control red force (enemy) roles in a training scenario. Use of these technologies ensures that virtual red forces do not tie up a more costly, high-fidelity simulator while still enabling trainees to fly against a human pilot who can most realistically mimic threat tactics and respond as the scenario unfolds. Additionally, game engines such as CryEngine® can be utilized to enable dynamic control of simulated friendly or red force ground vehicles within air-to-ground training scenarios.

Our recent efforts have included the incorporation of COTS game engines into networked training environments. One such effort was undertaken as part of a large-scale distributed training system display at the 2010 Interservice/Industry Training Simulation and Education Conference (I/ITSEC). A second effort involves the linking of multiple off-the-shelf technologies for a three-person training scenario for an upcoming training effectiveness study (Winner & Voge, 2011). Through these efforts we have identified a number of evaluation criteria relating to the selection of visual generation and simulation and game control for use in Live, Virtual, and Constructive (LVC) training environments. The purpose of this paper is to communicate our initial evaluation criteria related to the incorporation of game engines within currently existing USAF training environments. Minimum and preferred criteria, which effectively limit or enable game engines to be incorporated into simulated training environments, are specified when possible. In order to provide context for discussion of these evaluation criteria, we also summarize the overarching motivations relating to our use of modeling and simulation and game engines.

DEFINITION OF A GAME ENGINE

Before we outline our evaluation criteria, it is important to acknowledge that a game engine is any software that abstracts the process of creating visuals and input to a video game (Ward, 2008). A game engine is not a game in and of itself.

Rather it is a tool through which interactive real-time simulations can be created. What is the difference between a simulation and video game? In a broad sense, the only difference between the two is their purpose. They are both real time programs receiving input from a user and processing responding output. We believe that both may be used in concert to provide effective training for USAF training audiences.

MOTIVATION FOR THE USE OF GAME ENGINES

Our interest in game engines is motivated by the instructional needs of the USAF. The use of modeling and simulation for training individual and team-level skills for the USAF is not a new venture. However, game engines hold great promise with regard to instruction due to their controllability, high end visuals, inherently deployable nature, and ability to simulate real world events convincingly in real time.

Modeling and simulation and game engines enable training capabilities like those discussed previously. At a higher level, these technologies can support instructional methods specified by learning theory, such as providing virtual learning environments that enable individuals to work in realistic team settings with necessary tools and information sources to practice the problem solving skills required to successfully complete their mission (Wilson, 1996). A review of the relevant theory and literature related to instructional theory is outside of the scope of this paper. It is important, however, to summarize several of the concepts guiding the selection and use of modeling and simulation for research within our laboratory.

The Mission Essential Competency (MEC) process provides the foundation for many of our efforts. The MECs are "higher-order individual, team, and inter-team competencies that a fully prepared pilot, crew, flight, operator, or team requires for successful mission completion under adverse conditions and in a non-permissive environment" (Colegrove & Alliger, 2002). The outcomes of the MEC process conducted for a given training audience provide the foundation for the development of learning objectives and measures of performance and are utilized in the assessment of training capabilities for various simulation systems. Both scenario design based on learning objectives and performance feedback are essential for instruction. Although it is possible to have a virtual environment that does not enable feedback (Melham et al., 2009; Hays, 2006), feedback is essential for learning (Singer & Howey, 2009). This focus on providing trainees with feedback, along with the need to evaluate the effectiveness of our training systems (Kirkpatrick, 1976; 1996), are the motivations behind our current efforts to automatically conduct performance assessment in gaming environments (Winner & Voge, 2011).

Our use of game engines is motivated by the need to provide cost-effective, instructionally-sound training capabilities, yet we acknowledge that these engines have evolved under vastly different motivations, namely to provide entertaining and profitable games to large audiences. Accordingly, considerable effort is required to deliver scenarios based on specified learning objectives, deliver feedback to trainees, and to perform automated performance measurement. Although we cannot expect a commercial game engine to come equipped with these mission-specific components, we can convey some evaluation criteria relevant to implementing game engines within the context of the Air Force's currently existing simulation and training systems. The remainder of this paper will focus on these criteria.

EVALUATION CRITERIA

The evaluation criteria outlined in this section evolved through our experience incorporating game engines into networked training environments. We acknowledge that these criteria are unique to our purpose, which is to develop and advance effective training capabilities for warfighters to ensure that they are combat mission ready. The evaluation criteria listed are preferred capabilities that an engine should either posses, or should be capable of modification to adequately address these criteria.

Data Availability

Within this framework of government simulation, there are four common intercommunication protocols: Distributed Interactive Simulation (DIS), High Level Architecture (HLA), Test and Training Enabling Architecture (TENA), and Common Image Generator Interface (CIGI). It is not important that a game engine implements these protocols. Rather, it is important that game engines expose certain information to enable users to effectively join a game engine into a scenario. Some examples of the data include: position, velocity, rotation, velocity of rotation, acceleration, entity health, and height of terrain for a given x,y location. Location data for both shot origin and destination are needed. Ideally, building position would also be available. Additionally, status variables, such as whether an entity is smoking or on fire, whether a player is crouching, prone, or standing, and whether a vehicle has a running engine, would ideally be available. The availability of this data has implications for the feasibility of joining a game engine into a distributed simulation and, from an instructional perspective, enables or hinders automated performance measurement, the delivery of scenario feedback and after action review to trainees.

3D Model Correlation

In distributed simulation and training environments, multiple players have eyes on the same geographic area but may be looking at that area through different simulation or game-based systems (e.g., M2DART, CryEngine®). Players, regardless of game or simulation system, who see an entity (e.g. truck) as it travels across the terrain, should see the same model (e.g. white, flatbed truck). Accordingly, there is a need to easily convert OpenFlight models into a format to be imported into any system or game engine used. It is important to note that engineering teams in this domain often lack full-time access to artists with the appropriate skill set. The ability to use readily available models, especially OpenFlight models, helps to address this issue.

Ground Database Integration

Digital Terrain Elevation Data (DTED) file format for digital terrain elevation information is commonly used in distributed simulation and training environments. A game engine must be capable of instantiating real world terrain based on this information and allow for the conversion of internal coordinate systems to real world coordinate systems. These databases exist at different levels of resolution. High resolution databases can be correlated to a higher level of accuracy. Also, the higher the resolution, the easier it is to present the appropriate level of realism with respect to the resolution of other models depicted in the simulated world. Ideally, scenarios would not contain terrain at a significantly lower resolution than that of the building models or objects within the scene.

In addition to resolution requirements already discussed, the training scenarios we utilize often require satellite imagery to be overlaid onto the terrain. In an ideal world, game engines would have the ability to enhance the visual appearance of the imagery to account for its low resolution. As a pilot in a training scenario decreases their altitude, we would hope that the quality of the terrain visuals would not decrease.

Visual Detail

Depending on the selection of operating systems and other factors, game engines generally support either OpenGL or Direct3D (DirectX). Ideally, game engines would support both options. The use of DirectX 11 enables advanced rendering and lighting features (e.g., tessellation and per pixel lighting) which may assist in providing highly immersive training environments. The precise aims of the training will determine how much visual detail is required; however, engines which support both OpenGL and Direct3D would provide the most flexibility for training purposes.

The availability of an API abstraction layer within the game engine is of far greater importance than which type of API is used. The abstraction layer allows for control over the visuals of the game engine as it is running, and greatly increases the ability to modify a game into an effective training tool. For example, if one attempted to modify a flight simulator game to function as the sensor view for a Remotely Piloted Aircraft (RPA), the only way to modify the on-screen visuals would be at the API level, either directly or through the game's abstraction layer.

Animation Control

The use of network protocols such as DIS and HLA has implications for the smoothing ability requirements for game engines used within these contexts. These protocols may not give true location information for specific models at a refresh rate as high as that of game engines. Accordingly, the location updates received by the engine may result in a "jumpy" movement of the model. Ideally, game engines would provide a mechanism for dealing with this issue.

Specialized Visual Attributes

A number of specialized visual attributes are important for training scenarios for various platforms. For example, the ability to visualize infrared (IR) and night vision capabilities accurately is an important component for RPA sensor operator training. While IR visual modes can be imitated by changing overlay colors on the display screen, accurate IR and night vision modes require an engine to take into consideration certain environmental variables as well as material encoding aspects of models in the simulation. The ability to implement IR and night vision modes in a realistic manner greatly expands the training capability of a game engine. Shadows are also meaningful in this training focus area and ideally would be rendered at all viewing distances. The lack of particle effects (e.g., dust trail behind a moving vehicle, smoke, flames) takes away from the usefulness of an engine for training. Like shadows, particle effects provide meaningful information upon which some warfighters rely. Ideally, game engines would accommodate the visual attributes outlined above as well as any particle effects specified in the DIS protocol.

Physics Engine

Most game engines are equipped with a physics engine which provides realistic simulation of physical systems within the gaming environment. Collision detection and collision response, for example, are necessary to accurately simulate the interaction between players, objects, and the gaming environment itself. In the real-world, a soldier could not walk through the solid wall of a building, nor would a bullet or larger munition be able to escape the effects of gravity on the projectile's flight path, etc. The physics capabilities in a game engine ensure that the same is true in the gaming environment. To be considered a good candidate for use in military simulation and training, a game engine should be able to correctly interpret and simulate physics-bound behavior given *only the position and velocity of entities*. Ideally, the less information needed to translate the physics-based information from the shared DIS environment into a candidate game engine, the better. Essentially, a game engine should be able to interpolate information from the DIS environment with minimal external engineering work required.

Artificial Intelligence (AI) Capability

Artificial intelligence (AI) capabilities within gaming technologies are key to enhancing the training environment and improving training effectiveness. Computer-generated entities which respond appropriately as a scenario unfold can serve as either red (enemy) forces or blue (friendly) forces, or as neutrals who simply complicate or add dimension to a particular scenario. To be considered a candidate for use in military training, a game engine must possess AI capabilities, those AI capabilities must be easy to implement, and AI behavior must be reasonably predictable.

Scenario Design

In the military training arena, creation of realistic scenarios is of the utmost importance. In order to effectively equip warfighters to perform in their operational/combat roles, scenarios must reinforce the concepts and tasks which correlate to that role's required competencies (Symons, France, Bell & Bennett, 2006). With regard to selection of game engines, an engine should provide user-friendly tools (e.g. flowcharts) for scenario editing and management. Within the military training community, those who create the scenarios are often experts in the training field but may not possess software engineering expertise. Ideally, an engine's scripting capability would allow users to set specific parameters for entity movement and timing within a scenario, and have a mechanism through which to repeat or alter a scenario for a given trainee of group of trainees as needed.

Audio

Audio capabilities of game engines are an important factor in realistic training systems. The background audio of vehicle engines and munitions, for example, provides valuable feedback to the user. It is important that the audio capabilities accurately represent directional audio and sound degradation. The ability to easily modify audio attributes is important in that it allows custom sounds to be easily added to the system. Ideally, a game engine used in military training would also support the DIS radio standard.

Internet Dependency and Digital Rights Management (DRM)

Understandably, game engines often require internet connectivity to complete updates, provide access to shared resources, and for digital rights management functionality. This is a significant problem for government entities that do not have constant Internet access due to security restrictions. Engines requiring an internet connection to authenticate for each use are especially problematic. Government networks are invariably subject to heightened security restrictions which may render the use of internet dependent game engines impossible on these networks.

Cost

Among the primary advantages of game-based training for military applications is the ability to effectively train personnel without the costs incurred by pulling personnel from their day-to-day duties and paying travel expenses for trainees to participate in live and simulated training exercises in various locations. As such, cost is an important criterion in the selection of gaming technologies. While the source code in an Open Source game engine is available to the public free of charge, a developmental license for another game engine may range into the tens of thousands. A high cost developmental license is not only cost prohibitive, but effectively negates one of the key advantages of utilizing game engines for simulation-based training.

Supportability

Supportability refers to the amount of technical information and aid available to users and secondary developers through the primary developer and/or user community. Web forums and communities of practice are examples of such support. Because Open Source projects are widely available at no cost, they tend to have robust user communities that make changes and updates to the system while a smaller faction of the user group manages the project.

Supportability is also related to the openness of a particular game engine. In closed projects, changes can be made only by working with the developer, which can prove a slow and expensive process. The downfall of an open engine, however, is that such an engine is unlikely to be at the forefront of gaming development without a company to back that development with capital. As a result, some open engines may lack modern features that enable ease of use. A balance in supportability is found with an engine that is highly controllable while also being a product supported by a major company. When selecting an engine, it is important to balance cost, time, technology needs, controllability needs, and the desire for commercial entity involvement.

License Restrictions

License restrictions are a significant factor related to custom application of game engines within simulation and training environments. Although much of this work is being conducted for government entities, many of those performing the work are developing custom software for their defense contracting employers. Accordingly, the selection of an engine will be constrained by the license restrictions for the engine, even though it has nothing to do with the scenario generation capabilities the engine can offer.

CONCLUSIONS

Games and gaming technology constitute a burgeoning field of research and are proving to be a valuable tool in effective training both within and outside of the military context. In this paper we outlined our motivations for use of modeling and simulation and provided evaluation criteria for incorporation of game engines into pre-existing Air Force distributed mission-training systems. Data availability, 3D model correlation, and ground database integration are primary considerations. If a game engine does not accommodate these criteria as outlined above, it will most likely not be considered for incorporation into Air Force training systems as it would limit our ability to simulate a real world environment. The remaining criteria add significant value to the training environment and help to distinguish game engines from one another. By outlining these criteria, we hope to promote dialogue with game engine developers regarding the capabilities that make game engines viable options for use in the Air Force distributed mission training environment.

REFERENCES

Castel, A. D., Pratt, J., and Drummond, E. (2005). The effects of action video game experience on the time course of inhibition of return and the efficiency of visual search. *Acta Psychologica, 119,* 217-230.

Chatham, R. E. (2011). After the revolution: Game-informed training in the U.S. Military. In S. Tobias & J. D. Fletcher (Eds.) *Computer Games and Instruction (pp. 73-99).* Charolette, NC: Information Age Publishing, Inc.

Clarke, T. L. (1995). Distributed interactive simulation systems for simulation and training in the aerospace environment. *Proceedings of the Conference for the Society of Photo-Optical Instrumentation Engineers (Critical Reviews of Optical Science and Technology, Orlando, FL*, Apr 19-20, 1995.

Colegrove, C. M., & Alliger, G. M. (2002). Mission Essential Competencies: Defining combat mission readiness in a novel way. Paper presented at the NATO RTO Studies, Analysis and Simulation Panel (SAS) Symposium. Brussels, Belgium.

Green, C. S. & Bavelier, D. (2007). Action-video-game experience alters the spatial resolution of vision. *Psychological Science, 18(1),* 88-94.

Green, C. S. & Bavelier, D. (2006). Effect of action video games on the spatial distribution of visuospatial attention. *Journal of Experimental Psychology: Human Perception and Performance, 32(6),* 1465-1478.

Green, C. S. & Bavelier, D. (2003). Action video game modifies visual selective attention. *Nature, 423,* 534-537.

Hays, R. T. (2006). *The science of learning: A systems theory approach.* Boca Raton, FL: Brown Walker Press.

Hussain, T. S., Weil, S. A., Brunye´, T., Sidman, J., Ferguson, W., & Alexander, A., (2007). Eliciting and evaluating teamwork within a multi-player game-based training environment. In H. F. O'Neil and R. S. Perez (Eds.) *Computer Games and Team and Individual Learning.* Elsevier Ltd.

Kirkpatrick, D. (1976). Evaluation of training. In R. L. Craid (Ed.), *Training and development handbook: A guide to human resources development. New York, NY: McGraw-Hill.*

Kirkpatrick, D (1996). Revisiting Kirkpatrick's fourlevel model. *Training and Development, 1,* 54-57.

Li, R., Polat, U., Makous, W., and Bavelier, D. (2009). Enhancing the contrast sensitivity function through action video game training. *Natural Neuroscience, 12*, 549-551.

Milham, L., Carroll, M. B., Stanney, K., & Becker, W. (2009). Training Systems Requirements Analysis. In D. Schmorrow, J. Cohn & D. Nicholson (Eds.), *The PSI Handbook of Virtual Environments for Training and Education: Developments for the Military and Beyond. Volume 2: VE Components and Training Technologies* (pp 165-192). Westport, CT: Praeger Security International.

McKinley, R. A., McIntire, L. K., & Funke, M. A. (2011). Operator selection for unmanned aerial systems: Comparing video game players and pilots. *Aviation, Space, and Envrionmental Midicine, 82(6),* 635-642.

Pohl, A. J., & Walker, J. C. (2010). Leveraging game technology: When are games the answer? *Military Simulation & Training Magazine, 2/2010,* 22-26.

Singer, M. & Howey, A. (2009). Enhancing virtual environments to support training. In D. Schmorrow, J. Cohn & D. Nicholsons (Eds.), *The PSI Handbook of Virtual Environments for Training and Education: Developments for the Military and Beyond. Volume 2: VE Components and Training Technologies* (pp 407-421). Westport, CT: Praeger Security International.

Symons, S., France, M., Bell, J., & Bennett, W., Jr. (2005). Linking knowledge and skills to Mission Essential Competency-based syllabus development for Distributed Mission Operations (AFRL-HE-AZ-TR-2006-0041, ADA453737). Mesa, AZ: Air Force Research Laboratory, Warfighter Readiness Research Division.

Ward, Jeff (2008). What Is a Game Engine? *Retrieved on 24 June 2011 from http://www.gamecareerguide.com/features/529/what_is_a_game_.php.*

Wilson, B. (1996). What is a constructivst learning environment? In B. Wilson (Ed.), *Constructivist learning environments: Case studies in instructional design.* Englewood Cliffs, NJ: Educational Technology Publications, Inc.

Winner, J. L., & Voge, D. J. (2011). Gaming Research Integration for Learning Laboratory (GRILL) Performance Measurement Workshop. Mesa, AZ: Air Force Research Laboratory, Warfighter Readiness Research Division.

AUTHOR BIOGRAPHY

JENNIFER WINNER received her B.A. in Psychology from Wright State University and an M.S. in Applied Psychology from Arizona State University. Jennifer joined Lumir Research Institute, Inc. in 2008, focusing on the measurement of team performance in simulation and training environments.

STEPHEN NELSON received his Bachelor's degree in Game and Simulation Programming from DeVry University. Stephen has supported the Gaming Research Integration for Learning Laboratory (GRILL) since 2009.

2LT REBECCA BURDITT studied Behavioral Science at the United States Air Force Academy. Since commissioning in 2010, she has worked in support of the mission of the 711th Human Performance Wing, Human Effectiveness Directorate, Warfighter Readiness Research Division. 2Lt. Burditt is the program lead for the GRILL.

CAPT ADAM POHL studied electrical engineering at Embry-Riddle Aeronautical University while on a ROTC scholarship. After commissioning in 2006 he attended the Air Force Institute of Technology where he received his Masters in Computer Engineering. He has been a system engineer for the GRILL since 2008.

An Interactive Policy Simulator for Urban Dynamics

Terry Lyons

Dr. Jim Duggan

College of Engineering & Informatics,

National University of Ireland, Galway.

E-mail: {t.lyons4|james.duggan}@nuigalway.ie

KEYWORDS

Urban Dynamics, simulation, stock and flow, model

ABSTRACT

Urban Dynamics is a field of study which applies mathematical modelling techniques to the system of a city in order to quantitatively represent and analyse its progression over time. The simulation of this model allows stakeholders in the city to investigate the interactions of the city's components and the consequences of decisions on the city's advancement.

This project builds upon established research in this field by creating a software system that simulates an interactive Urban Dynamics Model, and presenting it as a learning tool in the form of an educational game. User's decisions drive the simulation, affecting the game's outcome, with the city either prospering or declining.

INTRODUCTION

The world around us is filled with complexities, with innumerable macro and micro systems that define it. As man has advanced, he has created his own systems, introducing further complexity to the natural world. These dynamic systems progress and evolve over the course of time, always changing based on the processes and interactions that drive them. The pursuit of understanding the world around us is the impetus for much of our scientific research. By dividing the indefinitely large and detailed universe into manageable, yet non-trivial systems, we can hope to gain some insight into the natural and man-made processes of our world. Systems Dynamics is a field of computer science which seeks to provide insight into such complex interacting systems, through formal mathematical simulation and quantitative analysis. Originally developed by Jay Forrester (*Forrester 1958, 1961*) for analysis of commercial dynamics, it has been more recently applied in a range of other human systems including business (*Sterman 2000*) and epidemiology (*Potash et al. 2003, Tebbens et al. 2009*).

Urban Dynamics (*Forrester 1969)* is an area of research within the field of System Dynamics in which study is conducted into the workings of an urban area. A city is a complex dynamic system that contains many heterogeneous, interacting components including population, industry and natural resources, which contribute to the working of the city as a whole. Over the course of its lifetime, a city will go through phases of growth and expansion, maturation and decay. As it advances, each of its processes will either promote or hamper the evolution of the urban area as a whole, encouraging its improvement, or forcing its decline.

The overall goal of this project is to create a flexible Urban Dynamics research and education simulation tool as a basis for exploring complex multi-sector urban systems. This should provide an understanding of a city's workings and allow for the assessment of various policy decisions, such as land zoning, natural resource management and water conservation. It is intended that the software system would support experimentation and learning about Urban Dynamics through the mode of an interactive computer game front-end. The project goal is accomplishedthrough the completion of three principal objectives (*Fig.1*):

1. Development of a mathematical Solving Engine for the simulation of Systems Dynamics models.
2. Design of an aggregate Urban Model, which represents a city's interactions and processes
3. Creation of an interactive game-based visualisation of the Urban Model, allowing users to introduce change into the system and observe its consequences.

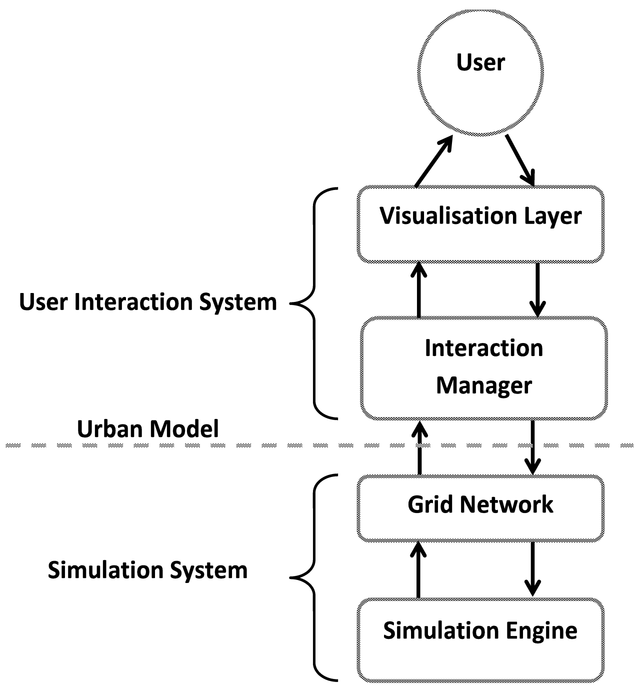


Figure 1: Overview of system Architecture

CONTRIBUTIONS

This system allows users to access a powerful and complex System Dynamics Simulation Engine through the intuitive and user-friendly interface of a game. Through this management game interface, those new to Urban Dynamics can interact with and observe the urban area as it evolves, without having to understand the underlying mathematics. Thus, this system can act as a learning environment and teaching tool.

Those who are experienced with Urban Dynamics may use the system to declare and simulate their own Urban Models, accessing the numeric results while at the same time being able to see their Model visualised in real-time. Through the game interface, they may make policy decisions and observe the impact on their urban setting. Thus, this system is also a research tool,providing policy-makers and researchers with an environment to simulate systems and plan decisions.

SYSTEM TECHNICAL DETAILS

Solving Engine

The computational foundation of this project is the Mathematical Solving Engine, which contains all the logic to solve a system of complex Stock and Flow Systems Dynamics equations. It includes a formal representation of Stocks and Flows, the algorithms for interpreting their values and equations, as well as providing for the dynamic solving of these at each step of a simulation run. An important feature of the engine is the *interaction* between Stocks and Flows, with the value of a Stock based on the summation of its Flows. The Engine uses Euler's Algorithm to solve the system of equations and provides for the communication between Stocks and Flows. This initiates Stocks and Flows, and evaluates all Stocks based on their corresponding In- and Out-Flows at each step in the simulation using Euler's Integration (*Equation 1*); it then solves Flows using the newly-calculated Stock values.

$$S_{t+dt} = S_t + dt*(I_t - O_t)$$

Equation 1: Euler's Integration

The Engine also contains an Extensible Operator Set, which allows for the definition of additional mathematical functions, such as Min, Max and Power Operators. As Stocks and Flows are solved as continuous variables, a decision was made to include a separate Solver component to handle discrete values. This discrete solver is based upon probabilistic Markov Chains. This additional functionality allows extra complexity to be added to the System Dynamics Model, with the Markov States able to interface with and alter the deterministic outcome of the Stock and Flow Model. As we shall see later, Markov Chains are used in the completed Urban Model to simulate discrete weather conditions, which affect certain components of the overall model.

Finally, the Engine disaggregates each sub-system of a System Dynamics Model, using a two-dimensional grid-based network to contain each Stock and Flow

Model that contributes to the simulation. As the Engine runs, it solves each Grid component independently. It then aggregates the results and shares the values with other Grid members, each cell acting as an individual that cooperates with all Grid members to advance the system as a whole.

User Interaction System

The Simulation Engine is a mathematical system that outputs numeric data, which needs to be interpreted and presented as ordered information for evaluation by a user of the software. For the purpose of this project, it was decided to create a game-based user interaction, which would interface with the Mathematical Engine and use its Simulation to drive the game. This method of presentation was chosen to provide the user with a learning tool to aid in the understanding of a complex Urban System. It visualises the data from the Mathematical Engine as an intuitive three-dimensional game world (*fig 2*), allowing the user to explore the Stocks of the Urban Model, represented as collections of 3D physical objects, such as buildings. This game-like presentation facilitates user input, inviting the user to make decisions, and allowing them to see the impact their choices make on the Urban Model.

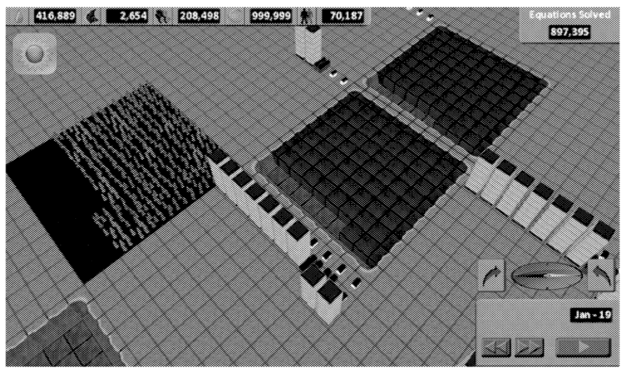


Figure 2: 3D game world, with building zone, forest, lake and additional data displayed on 2D GUI

Rather than running the whole Simulation and outputting the final results to the user, the User Interaction System provides interaction in real-time, with the user making changes and modifying the Urban Model in between steps of the Simulation. In this way, the user can very clearly see the consequences of their decisions and the evolution of the Urban Model over time. The System is divided into two layers: the Interaction Manager, which interfaces with the Engine and interprets its output, as well as allowing the user to modify the Urban Model; while the Visualisation Layer manages the Graphical User Interface, presenting all information onscreen in the 3D-world and 2D-menus, and handling user input.

While the Simulation Engine is of flexible and extensible design, it was necessary to tailor the Interaction System specifically to represent the Urban Model. Other more generic visualisations could have been used, such as network charts or Stock and Flow graphs, but the game-based presentation is a much more interactive experience that provides an involving way of understanding a complex Urban system. The 3D models of the game world were specifically designed to visualise the Stocks of the Urban Model, being mapped to each Stock in the Interaction Manager. The user's inputs adjust parameters within the Urban Model to alter the outcome of its Simulation, which in turn modifies what is being displayed onscreen. Therefore, while the Simulation Engine can solve any Stock and Flow Model, the User Interaction System is simply one possible way of interacting with it and visualising its data.

The Urban Model

The Urban Model is a complex, multi-sector Systems Dynamics Model, based on the project's requirement of simulating Urban Dynamics. Its design is an abstraction of interactions within a real-world urban setting, and draws from work in the field, including Forester's Urban Dynamics (*Forrester 1969*) and IBM's CityOne (*IBM 2010*), as well as management games such as SimCity, a city management game originally based on Systems Dynamics techniques (*Seabrook 2006*). The Model is split into numerous sub-systems, each a separate Systems Dynamics Model which represents a homogenous collection of urban interactions. Each of these sub-systems is in themselves independent, but interacts and drives the dynamics of the Urban Model as a whole. While the design seeks to replicate an urban setting, for the purposes of the project it is applied within the interactive environment of a game, and so certain considerations were made to facilitate this user interaction. Thus, each sub-system is target-driven, the goals of one Model advancing another. As the Model is simulated, the behaviour of the sub-systems is augmented with user input to seek these goals. In general, Stock and Flow Models are deterministic, with each Stock value based on the summation of its Flows over time. The Urban Model, however, includes a probabilistic Markov Chain-based Weather

System which, coupled with the unpredictability of user interaction, yields a non-deterministic Model.

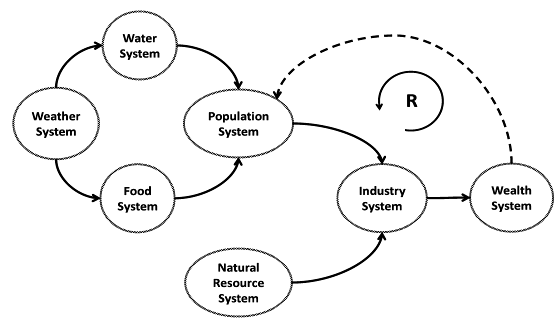


Figure 3: Overview of the Urban Model

In *Figure 3*, we see the architecture of the Urban Model at a high level, including each of its systems and the interactions they have with one another. The probabilistic Weather System drives the production of Water and Food, which are used in the growth of the Population System. Population and Natural Resources feed into the Industry System, which generates Wealth. The Wealth System creates a positive feedback loop, its value encouraging Population increase. The Population System is the central sub-system of the Urban Model, with the system's ultimate goal being to maximise Population. This is achieved by balancing the output of each of the other sub-systems, without overstepping the optimal value and hampering population growth. The inflow and outflow of Population in the Urban Model is determined by the system's Quality of Life.

Quality of Life (*QoL*) is a value between 0 and 100, and is a measure of the attractiveness of living in the city. High Quality of Life encourages immigration, while low Quality of Life promotes emigration. Its calculation is based on a metric taken from each of the Urban Model's systems, which are weighted equally in their impact on Quality of Life, shown in *Equation 2*. Each value is a fraction that measures how well their sub-system is encouraging growth. *WNF*, *FNF* and *GNF* take the actual consumption over the required consumption for the Water, Food and Industry Systems respectively. Each is directly proportional to Quality of Life, so that if any of these needs are not adequately fulfilled, *QoL* falls. *PD* is calculated within the Population System itself, and is a measure of the current space occupied by population divided by the total allowed space. It is impacted by the maximum sanctioned housing density, as discussed in the Population Model. *TI* is based on the Wealth System's current rate of tax, either low, medium or high. *PD* and *TI* are inversely proportional to *QoL*, so if eitherincreases, *QoL* drops. As each of these sub-systems interact with one other to form the Urban Model, encouraging optimal values for one metric may result in sub-optimal values for another metric, and so they must all be carefully monitored.

□□□=□□□*20+□□□*20+1−□□*20+□□□*20+1−□
□*20

WNF: Water Need Fulfilled
FNF: Food Need Fulfilled
PD: Population Density
GNF: Goods Need Fulfilled
TI: Tax Impact

Equation 2: Quality of Life calculation

TESTING

Each component is tested following an evaluation plan, which describes the testing methods and bounds of acceptance. The Solving Engine is stress tested to ensure mathematical correctness of all operations, while the SIR Model of infection was simulated within the Engine and benchmarked against an identical model in Vensim, to verify that it correctly solves Stock and Flow Models.

As an industry standard, Vensim was also used to test each of the Urban Model's sub-systems, which were declared and simulated to ensure correct behaviour. These sub-systems were then initialised with the same starting values within the Simulation Engine to and compared, to verify that the results were identical.

As the User Interaction System was being built, scalability tests were carried out to determine the optimal size of the World for the game. It was discovered that XNA could render a World Grid containing between 10,000 and 14,400 3D models before any noticeable drop in frame rate occurred. Thus, it was decided to set the size of the World Grid at 10,000 cells. With this limit imposed, the Simulation Engine's Network Grid size was set to 100, giving every Grid Model a region of 10x10 World Grid cells on which to be visualised.

CONCLUSIONS

The project's goal was to create a software system that would act as a simulation tool for Urban Dynamics research, providing an insight into the workings of a city by exploring multi-sector urban systems. To complete this goal, three principal objectives were defined:

1. Development of a mathematical Solving Engine for the simulation of Systems Dynamics models.
2. Design of an aggregate Urban Model, which represents a city's interactions and processes
3. Creation of an interactive visualisation of the Urban Model, allowing users to introduce change into the system and observe its consequences.

The final system acts as an intuitive, visual learning environment, allowing users to participate in the simulation of the Urban Model. Their interactions impact the evaluation of the Simulation Engine, and the corresponding changes that propagate through the Model's sub-systems are observable in the 3D world of the game as the simulation runs.

The desired flexibility in the Simulation Engine was achieved, being able to solve any equation specified in the structured Equation Syntax, and implementing the Extended Operator set, which provides the functionality to define a wide range of mathematical formulations. Thus, the Engine can solve any Stock and Flow Model, allowing it to be applied to a wide range of System Dynamics domains. Through the functionality of the Markov Chain Solver, probabilistic functions may be introduced to the Model, yielding a non-deterministic simulation. The inclusion of a two-dimensional Grid-based Simulation Network provides a structure for solving a collection of interacting System Dynamics Models, allowing for the cooperative simulation of the Urban Model's sectors. As set down in the evaluation plan, the logic of the Engine has been validated against the commercial simulation package, Vensim, verifying that it follows the correct procedure for solving Stock and Flow Models. Thus, this completedSimulationEngine is a benchmarked, flexible framework for use in System Dynamics research.

The final Urban Model is comprised of weather, food production, reservoir, natural resource, industry and wealth systems, all of which influence each other to drive a multi-sector conceptualisation of a city. Its design includes Auxiliary Variables which may be modified by the user, simulating policy decisions that impact the city's evolution. This user interaction, coupled with the stochastic Weather System, introduces non-determinism that affects the Model as a whole. While it is a high-level abstraction of a real city, the Urban Model never-the-less captures an urban area's important interactions to provide insight into the workings of a city. The behaviour of each of its systems has been verified in Vensim, where each Model was simulated, and the results analysed to ensure expected outcomes. The Urban Model is successfully implemented within the Simulation Engine's Network, allowing its independent sectors to influence each other and drive the advancement of the urban area as a whole.

The visualisation of the Urban Model was achieved by creating a computer game-based User Interaction Layer. This presents the urban area as a three-dimensional world, which the user can navigate around, observing the progression of the city as it simulates. Input through the game's menus allows the user to make policy decisions, such as introducing water recycling in reservoirs and changing the rate of felling in forests. They may also affect the physical landscape of the city, by zoning areas of land for residential, food production or industrial usage. These interactions impact the advancement of the urban area, the consequences of which the user can observe within the game world: refilling of reservoirs, felling of forests, and the construction on newly-zoned land over time (Figure 4). As with the other systems, the visualisation was tested following the evaluation plan. Scalability testing yielded an acceptable size for the game world which could visualise in real-time, while a comprehensive assessment of the system's allowable inputs was performed, verifying that all user interaction resulted in the intended response.

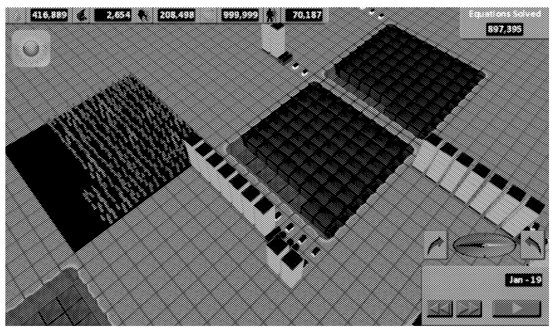


Figure 4: Finished visualisation, including a refilling reservoir, forest being felled, and a residential zone that is being built-up

FUTURE WORK

Further development of the system's features could produce a more streamlined, feature-rich System Dynamics tool, which could be adopted as a research aid. Currently, the system offers a generic Systems Dynamic Solving Engine which can solve a network of Models simultaneously, as well as an interactive urban system. Therefore, it can be used to simulate any Systems Dynamic Model, and as a learning environment for understanding urban interactions.

Within the Simulation Engine, further investigation of probabilistic mathematics could present alternative stochastic processes which could be augmented with the Markov Chain Solver. As the Simulation runs, each variable saves its value at every Step. Currently, this history of values is only held in program memory, and is lost when the program closes. Persistence of a Model's history would allow it to be saved and loaded at a later stage, as well as providing a complete numeric output of the Model's data. Additions to the Extended Operator set would create an exhaustive library of mathematical formulations.

Due to time constraints a planned Seasonal Weather System and multi-Model Natural Resource System were not implemented within the Urban Model. If further developed, the Markov Weather Model could also simulate the seasons, the transition probabilities of its weather conditions changing from season to season. It could also include rarer, more extreme weather conditions that would have a greater impact on the urban system. The implemented Natural Resource System uses only wood as a resource, but could be extended to include a Mining Model, representing the mining and refining of coal and ore. Other suggested systems include a trade and commerce system, a health and healthcare system, as well as extensions to Quality of Life such as pollution and entertainment. The inclusion of these and other sub-systems, coupled with more specific goal-seeking gameplay mechanics, could yield a more realistic urban simulation, as well as a more in-depth and challenging user interaction.

REFERENCES

Forrester, J. W. [1958] *"Industrial Dynamics-A Major Breakthrough for Decision Makers"* Harvard Business Review, Vol. 36, No. 4

Forrester, J. W. [1961] **"***Industrial Dynamics"* Pegasus Communications.

Forrester, J. W. [1969] "*Urban Dynamics"* Pegasus Communications.

IBM [2010] www.ibm.com/cityone

Potash, P. J., J. F. Heinbokel [2003] *"Modeling Human Behavior as a Factor in the Dynamics of an Outbreak of Pneumonic Plague"*presented at: 21st International System Dynamics Conference.

Seabrook, J. [2006] *"Game Master: Will Wright changed the concept of video games with the Sims. Can he do it again with Spore?"* The New Yorker, November 6, 2006.

Sterman, J. D. [2000] "Business Dynamics: *Systems Thinking and Modeling for a Complex World"* McGraw Hill.

Tebbens, R. J. D., K. M. Thompson [2009] "Priority Shifting and the Dynamics of Managing Eradicable Infectious Diseases" Management Science, Vol. 55, No. 4

Ventana Systems Inc. www.vensim.com

NASTEC

OPTIMIZATION OF PRODUCTION-MAINTENANCE POLICY BASED ON THE PRODUCTION RATE WITH A DISCRETE EVENT SIMULATION

Jérémie Schutz, Nidhal Rezg
Paul Verlaine University, LGIPM, Project COSTEAM INRIA Nancy Grand Est
Ile du Saulcy, 57045 Metz cedex 1,
FRANCE
E-mail: {schutz|rezg}@univ-metz.fr

KEYWORDS
Discrete event simulation, optimization, maintenance, production.

ABSTRACT

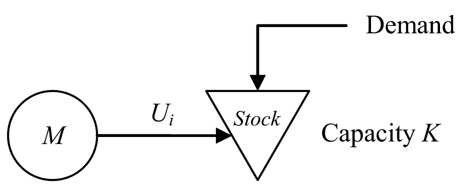
In this paper, we deal with the problem of optimization of production-maintenance policies. The manufacturing system consists of one machine producing a single type of product. These products supply a buffer stock of capacity *K* to meet a constant customer demand. Tthe degradation of the machine depends on the production rate. The aim of this paper is to find the optimal replacement periods, the optimal production rate and the optimal capacity of stock to optimize the objective function (minimizing operating costs). A discrete event simulation methodology is proposed to solve this problem in order to relax some constraints.

INTRODUCTION

At the beginning of industrialization in the 20th century, production and maintenance were two areas without interaction. At that time, demand was outstripping supply. It seemed inconceivable that preventive maintenance actions could disrupt production. Indeed, until the late 1990s, these two domains were studied separately.
In the production area, a lot of research deals the problems of job planning/scheduling. Generally, the main objective consists of determining the optimal scheduling that minimizes the makespan. From the "simple" flowshop problem (Johnson, 1954), new constraints are constantly added... For example, Lin (2001) considers a scheduling problem where sets of jobs are simultaneously available for processing in a no-wait two-machine flowshop. Wu et al. (2008) deal the scheduling problem with deteriorating jobs based on the linear deterioration models. To solve these problems, numerous algorithms and techniques have been developed. These problems can be solved by simple heuristic (Ruiz, 2005), meta-heuristics, without forgetting the exact methods.
In terms of maintenance, Barlow (1965) is considered a pioneer. He proposed different strategies for maintenance (age or block replacement). These strategies were based on the assumption that corrective actions were considered as minimal (As Bad As Old) and preventive actions as perfect (As Good As New). Among the major works, Pham (1996) proposed different models of imperfect maintenance to be closer to reality. Generally, for this maintenance area, works were focused on optimizing maintenance policies.
For about a decade, works that combine production and maintenance have emerged. On the one hand, these works are based on the problem of production with integration of maintenance planning. For example, (Sortrakul, 2005) proposes to perform preventive maintenance at the end of jobs. The integration between production plans and maintenance plans can be achieved along two main strategies (Benbouzid, 2006). These strategies are known as the sequential strategy (define a plan of production considered as a constraint to plan maintenance activities) and the integrated strategy (perform a joint and simultaneous scheduling between the tasks of maintenance and production). On the other hand, many research is based on maintenance policies considering production data. We can cite Chelbi et al. (2004) were the aim consists to determine the optimal period of replacement for an equipment and the value of buffer stock. When the equipment is new, the system builds up a buffer stock with a maximal production rate. The hedging point strategy is used to guarantee a continuous supply of the assembly line at a constant rate during repair and preventive maintenance actions. Recently, to optimize production and maintenance plans, Hajej et al. (2009) consider the production rate as a new variable. In this work, the authors determine simultaneously the optimal production rate and preventive maintenance intervals in the case of finite interval (Nakagawa, 2009).
Based on these observations, the objectives of our paper consist to determine the optimal period of replacement of an equipment, based on the optimal production rate and size of the buffer stock. Section 2 gives a description of the studied system. Section 3 presents the mathematical formulation. Section 4 presents the discrete event simulation used to solve this problem. A numerical example is used at the section 5 in order to illustrate our approach. Finally, section 6 gives the conclusion and the future works.

PROBLEM STATEMENT

In this paper, we consider a system of production composed of a single machine *M* producing a single product. The products are placed in a buffer stock used to meet the demands over time.



Figures 1: Production system

During the feed of stock, having a capacity K, M produces with the maximal production rate $U_{\max}$. After this delay, the production rate U_i is used.
The machine is subject to random failures, which require performing corrective maintenance actions. To reduce the corrective maintenance costs, preventive replacement is performed as $k.T$ (i.e. at T, $2T$, $3T$, ...). The buffer stock must satisfy, at least, the demand during the corrective maintenance activities following failures within the period of length T. During the preventive maintenance period, two cases may arise:

- The stock level is sufficient to fully satisfy demand.
- The stock level is insufficient to fully satisfy demand and there is a shortage of products.

The aim of this paper is to determine the optimal values of the three decision variables which minimize the average total cost by time unit. We search for the maximal capacity of stock (K*), the time between two preventive maintenance actions (T*) and the production rate (U_i*). When the production rate varies, the degradation of the system is different. Consequently, if the production rate increase, the number of failures during the horizon T is greater. To represent this effect, we have chosen to use the accelerated life model (ALM). In Martorell et al. (1999), the authors used this model to represent working conditions. For this model, a relationship is established between the baseline reliability function and a link function taking into account the working conditions. The usual mathematical model employed as link function is the loglinear. This model will be clarified in next section.

Notations

Throughout the paper, the following notations will be used:

$f(\cdot)$: Probability distribution function of time to failure for machine M.
$g_C(\cdot)$: Probability distribution function associated to corrective maintenance duration for machine M.
$g_P(\cdot)$: Probability density function associated to preventive maintenance duration for machine M.
MTBF: Mean time between failures of machine M.
MTTR: Mean time to repair of machine M.
Z_P : Mean time of preventive maintenance activity ($Z_P = \int_0^\infty t.\mathrm{d}G_P(t)$)
$U_{\max}$, U_i: Production rates (products per time unit).
d: demands per time unit.
C: Average total cost per time unit.
C_H: Holding cost of a product unit during a unit of time.
C_H^{TOT} : Total holding cost
C_S: Shortage cost of a product unit during a unit of time.
C_S^{TOT} : Total shortage cost
C_M: Total cost of maintenance actions.
C_{MC}: Corrective maintenance action cost.
C_{MP} : Preventive maintenance action cost.

MATHEMATICAL FORMULATION

To model the effect of the production rate, we have chosen to use the accelerated life model. The reliability function in accelerated model is given by:

$$R(t) = R_0\left(\psi(U_i)\cdot t\right) \qquad (1)$$

where $R_0(t)$ is a baseline reliability function and $\psi(U_i)$ is the link function, which depends on the production rate. Based on our problem, we chose to define the link function as follows:

$$\psi(U_i) = e^{\left(\alpha\cdot\frac{U_i-d}{U_{\max}}\right)} \qquad (2)$$

Where α is a coefficient which can be specific to each type of machine. Based on equation 2, if the machine does not produce, the reliability remains unchanged with time. However, if the production rate increases, the degradation process of the system is accelerated. For the ALM, other functions can be determined. The hazard and the density probability functions are respectively given by:

$$h(t) = \psi(U_i)\cdot h_0\left(\psi(U_i)\cdot t\right) \qquad (3)$$

$$f(t) = \psi(U_i)\cdot h_0\left(\psi(U_i)\cdot t\right)\cdot R_0\left(\psi(U_i)\cdot t\right) \qquad (4)$$

From equation 1, the variation of production rate has an effect on system reliability. Consequently, the *MTBF* and the number of failures during the period T vary following the production rate.
From Chelbi et al. (2004), the renewal function $M(T)$ is defined as the expected number of cycles (*MTBF*, *MTTR*) in [0; T]. Considering the effect of the production rate U_i, we can model our renewal function $M(T;U_i)$ by:

$$M(T,U_i) = \sum_{n=1}^{\infty} L^{(n)}(T,U_i) \qquad (5)$$

where $L^{(n)}(T;U_i)$ represents the nth convolution of $L(T;U_i)$, i.e.

$$L^{(n)}(T,U_i) = \int_0^T L^{(n-1)}(T-x,U_i)\mathrm{d}L(x,U_i) \qquad (6)$$

and $L(T)$ is the convolution product of the life time and repair time distributions,

$$L(T,U_i) = \int_0^T F(T-x,U_i)\mathrm{d}G_C(x) \qquad (7)$$

In this paper, the feed of stock is performed with the production rate $U_{\max}$. Consequently, the duration to reach the capacity K is short and we assume that the number of failures during this period is negligible. So, we define a function $\phi(H;U_i)$ which represents the number of failures during a period H, with the production rate U_i, based on the renewal function.
For our model, we search the optimal decision variables (T*, K*, U_i*) which minimize the total expected cost per time unit which includes inventory holding cost, shortage cost and maintenance cost.

Inventory holding cost

To compute the inventory holding cost, we have considered the five zones of the figure 2. When the production rate is equal to the demand during the period [A; T], the total cost of inventory holding is given by:

$$C_H^{TOT}(K,T,U_i)=C_S\cdot(Z_1+Z_2+Z_3+Z_4+Z_5) \qquad (8)$$

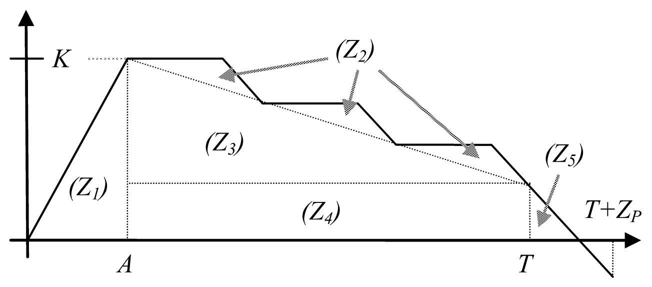


Figure 2: Evolution of the stock between two replacements when $U_i = d$

With:

$$Z_1=\frac{1}{2}\cdot\frac{K^2}{U_{\max}-d} \qquad (9)$$

$$Z_2=\phi(T-A,d)\cdot\begin{pmatrix}\frac{1}{2}\cdot(MTBF+MTTR)\cdot d\cdot MTTR\\ -\frac{1}{2}\cdot d\cdot MTTR^2\end{pmatrix} \qquad (10)$$
$$=\phi(T-A,d)\cdot\frac{1}{2}\cdot(MTBF+MTTR)\cdot d$$

$$Z_3=\left(T-\frac{K}{U_{\max}-d}\right)\cdot\left(\frac{1}{2}\cdot\phi(T-A,d)\cdot d\cdot MTTR\right) \qquad (11)$$

$$Z_4=\left(T-\frac{K}{U_{\max}-d}\right)\cdot(K-\phi(T-A,d)\cdot d\cdot MTTR) \qquad (12)$$

$$Z_5=\frac{1}{2}\cdot\left(\frac{K-\phi(T-A,d)\cdot MTTR^2}{d}\right) \qquad (13)$$

When the production rate is different to the demand during the period [A; T], we denote three possibilities. For a single cycle ($MTBF$ +$MTTR$), we have:

- Case A: The production rate is higher than the demand. During the $MTBF$, the inventory level increases with time but...
  - Case A_1: The demand during the $MTTR$ is higher than the production (less demand) during the $MTBF$ (Figure 3).
  - Case A_2: The demand during the $MTTR$ is lower than the production (less demand) during the $MTBF$ (Figure 4).
- Case B: The production rate is lower than the demand. During the $MTBF$, the inventory level decreases with time (Figure 5).

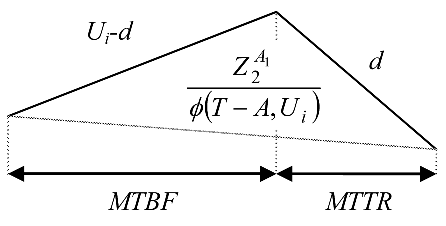


Figure 3: Zoom on the Z_2 area for the case A_1

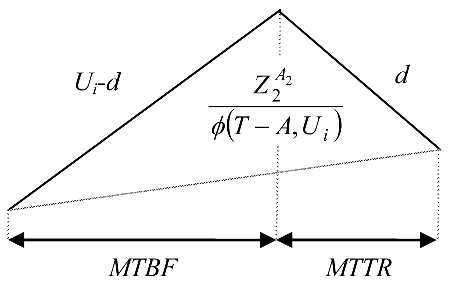


Figure 4: Zoom on the Z_2 area for the case A_2

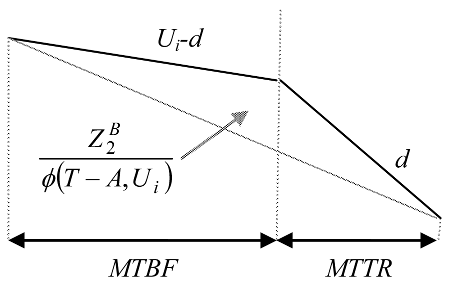


Figure 5: Zoom on the Z_2 area for the case B

The total cost of inventory holding is given by:

$$C_H^{TOT}(K,T,U_i)=C_S\cdot(Z_1+Z_2'+Z_3'+Z_4'+Z_5') \qquad (14)$$

$$Z_2' = \phi(T-A,U_i)\cdot \left[\frac{1}{2}\cdot MTBF\cdot\left(MTTR\cdot d + \left(MTTR\cdot d - MTBF\cdot(U_i-d)\right)\right) + \frac{1}{2}\cdot MTTR\cdot\left(MTTR\cdot d\right)\cdot + \left(MTTR\cdot d - MTBF\cdot(U_i-d)\right)\right] = \phi(T-A,U_i)\cdot\frac{1}{2}\cdot\left(MTBF\cdot MTTR\cdot U_i\right) \quad (15)$$

$$Z_3' = \left(T-\frac{K}{U_{\max}-d}\right)\cdot\left(\frac{1}{2}\cdot\phi(T-A,U_i)\cdot \left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right) \quad (16)$$

$$Z_4' = \left(T-\frac{K}{U_{\max}-d}\right)\cdot\left(K-\phi(T-A,U_i)\cdot \left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right) \quad (17)$$

$$Z_5' = \frac{1}{2}\cdot\left(\frac{\left(K-\phi(T-A,U_i)\right.}{d}\cdot \left.\left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right)^2\right) \quad (18)$$

Shortage cost

The total cost of shortage corresponds to the product of the shortage cost per unit per time unit by the number of product which are not delivered to the customer. From the figure 2, we have:

$$C_S^{TOT}(K,T,U_i) = C_S\cdot\frac{1}{2\cdot d}\cdot\left(d\cdot Z_P - \left(K-\left(\phi(T-A,d)\cdot MTTR\cdot d\right)\right)\right)^2 \quad (19)$$

For each case defined previously, we obtain the same result:

$$C_S^{TOT}(K,T,U_i) = C_S\cdot\frac{1}{2\cdot d}\cdot\left(d\cdot Z_P - \left(K - \left(\phi(T-A,U_i)\cdot\left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right)\right)\right)^2 \quad (20)$$

Maintenance cost

For the maintenance strategy, the total cost of maintenance is given by:

$$C_M(K,T,U_i) = C_{MC}\cdot\phi(T-A,U_i) + C_{MP} \quad (21)$$

Average total cost

For this model, the average total cost by time unit is given by:

$$C(K,T,U_i) = \frac{1}{T+Z_P}\cdot\Bigg(C_H\cdot\left(\begin{array}{l} \frac{1}{2}\cdot\frac{K^2}{U_{\max}-d} \\ +\phi(T-A,U_i)\cdot\frac{1}{2}\cdot\left(MTBF\cdot MTTR\cdot U_i\right) \\ +\left(T-\frac{K}{U_{\max}-d}\right)\cdot\left(\frac{1}{2}\cdot\phi(T-A,U_i)\cdot \left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right) \\ +\left(T-\frac{K}{U_{\max}-d}\right)\cdot\left(K-\phi(T-A,U_i)\cdot \left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right) \\ +\frac{1}{2}\cdot\left(\frac{\left(K-\phi(T-A,U_i)\right)}{d}\cdot \frac{\left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)^2}{d}\right) \end{array}\right) + C_S\cdot\frac{1}{2\cdot d}\cdot\left(d\cdot Z_P - \left(K - \left(\phi(T-A,U_i)\cdot\left(d\cdot MTTR - MTBF\cdot(U_i-d)\right)\right)\right)\right)^2 + C_{MC}\cdot\phi(T-A,U_i) + C_{MP}\Bigg) \quad (22)$$

To obtain the values of K^*, T^* and U_i^*, the necessary condition is that the partial derivative with respect to each decision variable must be zero.

$$\frac{\partial C(K,T,U_i)}{\partial K} = 0 \quad (23)$$

$$\frac{\partial C(K,T,U_i)}{\partial T} = 0 \quad (24)$$

$$\frac{\partial C(K,T,U_i)}{\partial U_i} = 0 \quad (25)$$

The computations are presented in Appendix A.
It is clear that the analytical resolution of this problem is difficult. Also, to minimize the average total cost, we use a discrete event simulation as defined in the next section. Furthermore, simulation offers other advantages. It becomes possible to relax constraints in order to approach the actual operation of the system. In the numerical section, we will relax the assumption of "no failure during the feed of stock".

DISCRETE EVENT SIMULATION

Our study is based on the discrete event simulation. To model our system, we must define the set of its states and the events that create transitions between these states...
The system is quite simple. It consists of a machine whose states correspond to "the machine is functional (R = 1)" or "the machine is in maintenance (R = 0)".
The events that create transitions between the states of the system and that describe its behavior are the following :

- Start of the corrective maintenace;
- End of the corrective maintenance;
- End of production of a product;
- Arrival of a demand

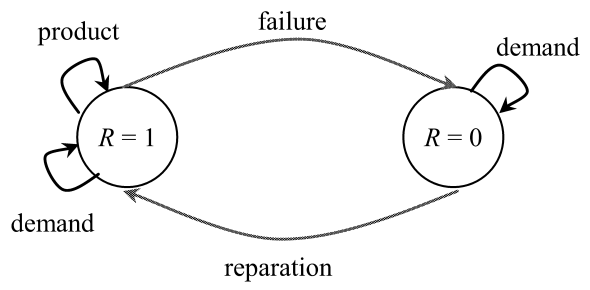


Figure 6: Finite-state machine

We can note that the arrival of a demand is not dependent on the state of the system.

Algorithm of simulation

The algorithm of simulation is described by the following flow chart:

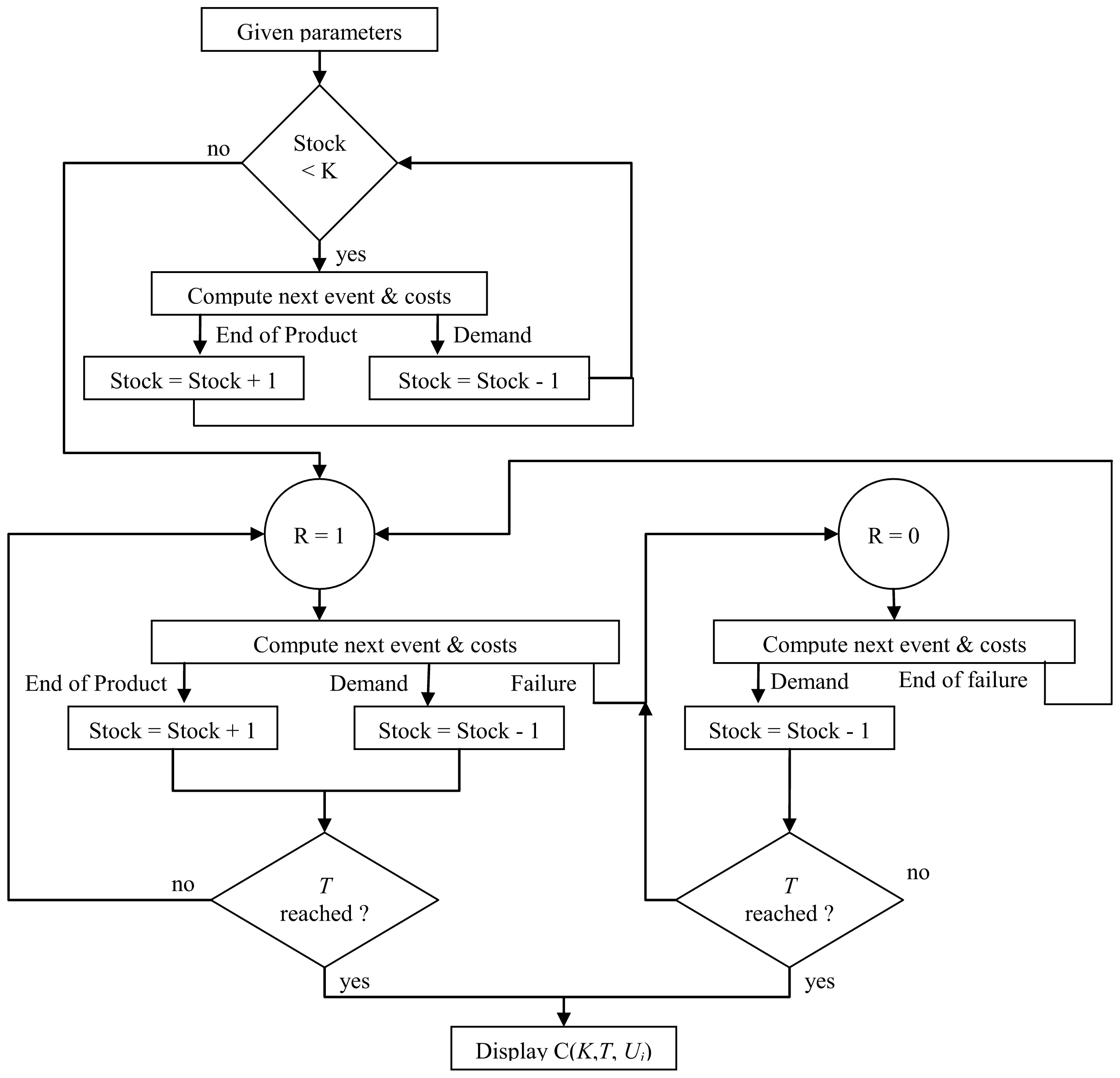


Figure 7: Diagram describing the algorithm of simulation.

The algorithm of simulation is given in appendix B.

NUMERICAL EXAMPLE

To illustrate the research work presented in this paper, numerical examples will be based on the following data:

$$F(t) = 1 - e^{-\left(\frac{t}{50}\right)^3}$$

$U_{\max}$ = 60 products/ tu (tu: time unit)
d = 20 products/tu
α = 0,4

$$G_C(t) = 1 - e^{-\frac{1}{5} \cdot t}$$

$$G_P(t) = 1 - e^{-\frac{1}{20} \cdot t} \text{ tu}$$

C_H = 0.25 mu/tu (mu: money unit)
C_S = 10 mu/tu
C_{MC} = 500 mu
C_{MP} = 2000 mu

For the simulation, we have chosen an increment of 10 tu for the variable *T*, 5 products for *K* and 5 products/tu for U_i.
The table 1 indicates the ten best results.

Table 1: Table of results

T	*K*	U_i	$C(K,T,U_i)$
130	20	25	78,05
130	25	25	78,14
130	30	25	78,30
130	35	25	78,54
140	20	25	76,47
140	25	25	76,99
140	30	25	77,58
140	35	25	78,23
150	20	25	77,17
150	25	25	78,07

From this table, we can ascertain there is not "one optimal value" because the average costs per time unit are quite close. Although the value of *T* was tested in the interval [20, 700] good solutions are obtained when *T* is around 130 – 150 tu. It is primarily a trade-off between holding costs and maintenance costs.
The variable *K* does not seem to have much effect. However, for these values (of *K*), we have no shortage. The differences between the various average costs per time unit correspond to the holding cost for an increment of 5 products during two preventive maintenances.
Concerning the production rate, we obtain 25 pr/tu as optimal value, which is slightly higher than demand. This increase of production rate compensates downtime due to corrective maintenances without requiring a large buffer stock.

When the variable *T* is fixed, th increase of the average total cost is linked to the production rate. This cost is is mainly due to the holding cost induced by the surplus of manufactured products.

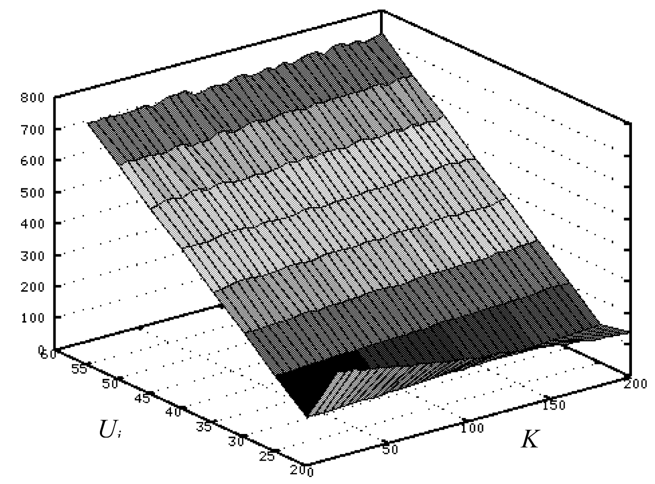


Figure 8: Average total cost per unit time with *T* = 140 tu.

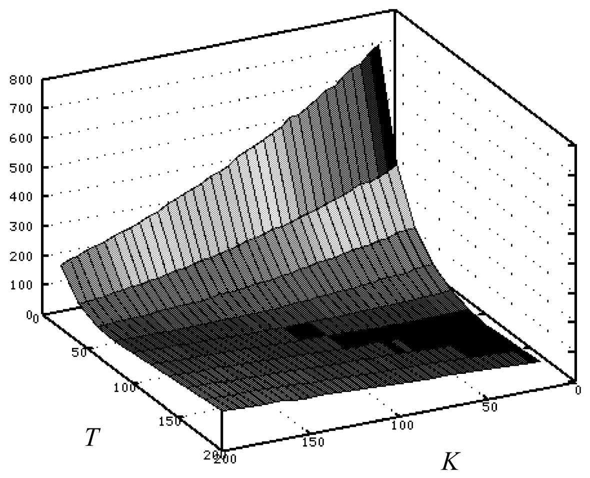


Figure 9: Average total cost per unit time with U_i = 25 product/tu.

CONCLUSION

This work can be seen as a continuation of works intiated by Chelbi et al. (2004). We added a new decision variable, namely production rate. Thanks to this new decision variable, the risk of shortage is minimized. Indeed, during the *MTBF*, the buffer stock increases to offset declines as a result of demands. However, the MTBF is linked to this decision variable. The risk of failures increases the higher of production rate. From the numerical example, we remark that it is favorable to produce with a slightly higher production rate compared to the demand.
In future work, it might be interresting to consider imperfect maintenances. In this case, the duration of maintenance actions would not be the same but would be based on efficiency factors.

REFERENCES

Barlow, R.E. and F. Proschan. 1965. Mathematical theory of reliability. John Wiley & Sons.

Benbouzid-Sitayeb, F., C. Varnier and N. Zerhouni. 2006. "Résolution du problème de l'ordonnancement conjoint production/maintenance par colonies de fourmis. " *Proceedings of the 6ème Conférence Francophone de MOdélisation et SIMulation (MOSIM'06)*, Rabat, Maroc.

Chelbi A, and D Ait-Kadi. 2004. "Analysis of a production1inventory system with randomly failing production unit submitted to regular preventive maintenance." *European Journal of Operational Research*, 156, 712–718.

Hajjej, Z., S. Dellagi and N. Rezg. 2009. "Optimization of a maintenance strategy with considering the influence of the production plan on the manufacturing system degradation." *13th IFAC Symposium on Information Control Problems In Manufacturing*, Moscow, Russia.

Johnson, S.M., 1954. "Optimal two- and three-stage production schedules with setup times included." *Naval Research Logistics Quarterly*, 1, 61-68.

Lin, B.M.T. and T.C.E. Cheng. 2001. "Batch scheduling in the no-wait two-machine flowshop to minimize the makespan." *Computers & Operations Research*, 28(7), 613-624.

Martorell, S., A. Sanchez and S. Vicente. 1999. "Age-dependent reliability model considering effects of maintenance and working conditions." Reliability Engineering and System Safety, 64(1), 19–31.

Nakagawa, T. and S. Mizutani. 2009. "A summary of maintenance policies for a finite interval". *Reliability Engineering and System Safety*, 94(1), 89-96.

Pham, H. and H. Wang. 1996. "Imperfect maintenance." *European Journal of Operational Research*, 94(3), 425–438

Ruiz, R. and C. Maroto. 2005. "A comprehensive review and evaluation of permutation flowshop heuristics". *European Journal of Operational Research*, 165, 474-494.

Sortrakul, N., H.L. Nachtmann and C.R. Cassady. 2005. "Genetic algorithms for integrated preventive maintenance planning and production scheduling for a single machine." *Computers in Industry*, 56(2), 161–168.

Wu, C.C, Y.R Shiau and W.C. Lee. 2008. "Single-machine group scheduling problems with deterioration consideration." *Computers & Operations Research*, 35(5), 1652- 1659.

APPENDIX A

$$\frac{\partial C(K,T,U_i)}{\partial T} = \left(\frac{1}{(T+Z_P)^2}\right)\cdot($$

$$C_H\cdot\left(\begin{array}{l} \frac{\partial\phi(T-A,U_i)}{\partial T}\cdot\frac{1}{2}\cdot(MTBF\cdot MTTR\cdot U_i) \\ +\left(\frac{\phi(T-A,U_i)}{2}+\left(\frac{T}{2}-\frac{K}{2\cdot(U_{\max}-d)}\right)\cdot\frac{\partial\phi(T-A,U_i)}{\partial T}\right) \\ \quad\cdot(d\cdot MTTR-MTBF\cdot(U_i-d)) \\ +T+\left(-\phi(T-A,U_i)+\left(\frac{K}{2\cdot(U_{\max}-d)}-T\right)\cdot\frac{\partial\phi(T-A,U_i)}{\partial T}\right) \\ \quad\cdot(d\cdot MTTR-MTBF\cdot(U_i-d)) \\ -\frac{1}{2}\cdot\left(\frac{\partial\phi(T-A,U_i)}{\partial T}\cdot\frac{(d\cdot MTTR-MTBF\cdot(U_i-d))^2}{d}\right)\end{array}\right)$$

$$+C_S\cdot\frac{1}{2\cdot d}\cdot(d\cdot Z_P-(K-$$
$$(\phi(T-A,U_i)\cdot(d\cdot MTTR-MTBF\cdot(U_i-d))))^2$$
$$\left.+C_{MC}\cdot\frac{\partial\phi(T-A,U_i)}{\partial T}\right)$$

$$-C_H\cdot\left(\begin{array}{l} \frac{1}{2}\cdot\frac{K^2}{U_{\max}-d} \\ +\phi(T-A,U_i)\cdot\frac{1}{2}\cdot(MTBF\cdot MTTR\cdot U_i) \\ +\left(T-\frac{K}{U_{\max}-d}\right)\cdot\left(\frac{1}{2}\cdot\phi(T-A,U_i)\cdot\right. \\ \qquad (d\cdot MTTR-MTBF\cdot(U_i-d))) \\ +\left(T-\frac{K}{U_{\max}-d}\right)\cdot(K-\phi(T-A,U_i)\cdot \\ \qquad (d\cdot MTTR-MTBF\cdot(U_i-d))) \\ +\frac{1}{2}\cdot\left(\frac{(K-\phi(T-A,U_i)}{d}\cdot\right. \\ \qquad \left.(d\cdot MTTR-MTBF\cdot(U_i-d)))^2\right)\end{array}\right)$$

$$-C_S\cdot\frac{1}{2\cdot d}\cdot(d\cdot Z_P-(K-$$
$$(\phi(T-A,U_i)\cdot(d\cdot MTTR-MTBF\cdot(U_i-d))))^2$$
$$-C_{MC}\cdot\phi(T-A,U_i)+C_{MP})$$
$$=0$$

$$\frac{\partial C(K,T,U_i)}{\partial K} = \frac{1}{T+Z_P}\cdot($$

$$C_H\cdot\left(\begin{array}{l} \frac{K}{U_{\max}-d} \\ +\left(-\frac{1}{U_{\max}-d}\right)\cdot\left(\frac{1}{2}\cdot\phi(T-A,U_i)\cdot\right. \\ \qquad (d\cdot MTTR-MTBF\cdot(U_i-d))) \\ +\left(T-\frac{(2\cdot K-\phi(T-A,U_i)\cdot}{U_{\max}-d}\right)\cdot \\ \qquad (d\cdot MTTR-MTBF\cdot(U_i-d))) \\ +\frac{1}{2}\cdot\left(\frac{(d\cdot MTTR-MTBF\cdot(U_i-d))^2}{d}\right)\end{array}\right)$$

$$\left.+\frac{C_S}{d}\cdot\left(\begin{array}{l} K-d\cdot Z_P \\ +\phi(T-A,U_i)\cdot(d\cdot MTTR-MTBF\cdot(U_i-d))\end{array}\right)\right)$$
$$=0$$

$$
\begin{aligned}
\frac{\partial C(K,T,U_i)}{\partial U_i} = {} & \left(\frac{1}{T+Z_P}\right)\cdot\Bigg(\\
& \frac{C_H\cdot\partial\left(\begin{array}{l}
\frac{1}{2}\cdot\frac{K^2}{U_{\max}-d} \\
+\phi(T-A,U_i)\cdot\frac{1}{2}\cdot(MTBF\cdot MTTR\cdot U_i) \\
+\left(T-\frac{K}{U_{\max}-d}\right)\cdot\Big(\frac{1}{2}\cdot\phi(T-A,U_i)\cdot \\
\qquad (d\cdot MTTR-MTBF\cdot(U_i-d))) \\
+\left(T-\frac{K}{U_{\max}-d}\right)\cdot\big(K-\phi(T-A,U_i)\cdot \\
\qquad (d\cdot MTTR-MTBF\cdot(U_i-d))) \\
+\frac{1}{2}\cdot\Bigg(\frac{(K-\phi(T-A,U_i))}{d}\cdot \\
\qquad (d\cdot MTTR-MTBF\cdot(U_i-d)))^2\Bigg)
\end{array}\right)}{\partial U_i} \\
& +C_S\cdot\frac{\partial}{\partial U_i}\Bigg(\frac{1}{2\cdot d}\cdot\big(d\cdot Z_P-\big(K- \\
& \qquad (\phi(T-A,U_i)\cdot(d\cdot MTTR-MTBF\cdot(U_i-d))))^2\Bigg) \\
& +C_{MC}\cdot\frac{\partial\phi(T-A,U_i)}{\partial U_i}\Bigg) \\
& -C_{MC}\cdot\phi(T-A,U_i)+C_{MP}\Big) \\
& =0
\end{aligned}
$$

APPENDIX B

The various inputs are given below:

- Failure law: $f(t)$
- Distribution of corrective maintenance time: $g_C(t)$
- Distribution of preventive maintenance time: $g_P(t)$
- Productin rate: $U_{\max}$, U_i
- Holding cost: C_S
- Shortage cost: C_P
- Maintenance costs: $C_{MC,}$ C_{MP}
- Demand: d

The output corresponds to the total expected cost per time unit (inventory holding cost, shortage cost and maintenance cost) for each values of (K,T, U_i)

The algorithm is given below:

```
I. INITIALIZATION
I.1 Initialize the effective clock of simulation        TTOT = 0
I.2 Initialize the buffer stock level                     NS = 0
I.3 Initialize the state of the equipment                  R = 1
I.4. Generate the TBF                                      v / f(t)
I.5 Do PT = $1/U_{max}$, dem = 1/d
II. BUILD UP A BUFFER STOCK
II.1 While TTOT < T
  II.1.1 While NS < K
    II.1.1 Do z = min(PT, dem)
    II.1.2 Do PT = PT – z, dem = dem – z
    II.1.4 If PT = 0 (end of a product)
      II.1.4.1 Do NS = NS + 1
      II.1.4.2 Do PT = $1/U_{max}$
    II.1.5 If dem = 0 (arrival of a demand)
      II.1.5.1 Do NS = NS - 1
      II.1.5.2 Do dem = 1/d
    II.1.6 Do $C_H^{TOT} = C_H^{TOT} + C_H \cdot NS \cdot z$
    II.1.7 Do TTOT = TTOT + z
  II.1.2 End while
  II.1.3 Break;
II.2 End while
III. IF R = 1
III.1 Do z = min(v, PT, dem)
III.2 Do v = v – z; PT = PT – z, dem = dem – z
III.3 If v = 0 (begin of failure)
  III.3.1 Do R = 0
  III.3.2 v / g(t) (generate TTR)
III.4 If PT = 0 (end of a product)
  III.4.1 Do NS = NS + 1
  III.4.2 Do PT = $1/U_i$
III.5 If dem = 0 (arrival of a demand)
  III.5.1 If NS = 0
    III.5.1.1 Do $C_S^{TOT} = C_S^{TOT} + C_S$
  III.5.2 Else
    III.5.2.1 Do NS = NS - 1
  III.5.3 Do dem = 1/d
III.6. Do $C_H^{TOT} = C_H^{TOT} + C_H \cdot NS \cdot z$
III.7. Do TTOT = TTOT + z
IV. IF R = 0
IV.1 Do z = min(v, dem)
IV.2 Do v = v – z; PT = PT – z
IV.3 If v = 0 (begin of production)
  IV.3.1 Do R = 1
  IV.3.2 v / f(t) (generate TBF)
IV.4 If dem = 0 (arrival of a demand)
  IV.4.1 If NS = 0
    IV.4.1.1 Do $C_S^{TOT} = C_S^{TOT} + C_S \cdot NS \cdot z$
  IV.4.2 Else
    IV.4.2.1 Do NS = NS - 1
  IV.4.3 Do dem = 1/d
IV.5. Do $C_H^{TOT} = C_H^{TOT} + C_H \cdot NS \cdot z$
IV.6. Do $C_M^{TOT} = C_M^{TOT} + C_{MC}$
IV.7. Do TTOT = TTOT + z
V. IF TTOT < T go to III.
VI. PREVENTIVE MAINTENANCE
VI.1. Do $\mu_P = Z_P$
VI.2. While $\mu_P > 0$
  VI.2.1 While NS > 0
    VI.2.1.1 Do z= dem
    VI.2.1.2 Do NS = NS – 1
    VI.2.1.3 Do $\mu_P = \mu_P - z$
    VI.2.1.4 Do dem = 1/d
    VI.2.1.5 Do $C_H^{TOT} = C_H^{TOT} + C_H \cdot NS \cdot z$
  VI.2.2 End while
  VI.2.3 Do z= dem
  VI.2.4.Do $C_S^{TOT} = C_S^{TOT} + C_S \cdot NS \cdot z$
  VI.2.5 Do $\mu_P = \mu_P - z$
VI.3 End while
VI.4 Do $C_M^{TOT} = C_M^{TOT} + C_{MP}$
VII. Display C = $(C_H^{TOT} + C_P^{TOT} + C_M^{TOT})/(T+Z_P)$
```

AUTHOR LISTING

AUTHOR LISTING