SIGRAD'05

The Annual SIGRAD Conference
Special Theme: Mobile Graphics
November 23-24, 2005
Lund, Sweden

Conference Proceedings

Edited by
Tomas Akenine-Möller

SIGRAD 2005

The Annual SIGRAD Conference
Special Theme – Mobile Graphics
November 23-24, 2005
Lund, Sweden

Conference Proceedings

Organized by

Svenska Föreningen för Grafisk Databehandling
and
Lund University

Edited by
Tomas Akenine-Möller

Published for
Svenska Föreningen för Grafisk Databehandling
(SIGRAD)
by Linköping University Electronic Press
Linköping, Sweden, 2005

Linköping Electronic Conference Proceedings, No. 16

Linköping University Electronic Press

Linköping, Sweden, 2005

ISBN 91-85457-86-8 (print)

ISSN 1650-3686 (print)

http://www.ep.liu.se/ecp/016/

ISSN 1650-3740 (online)

Print: E-husets tryckeri, Lunds universitet, 2005

Table of Contents

Preface

These proceedings contain the papers from the SIGRAD 2005 conference which was held on the 23rd and 24th of November in Lund, Sweden. The topic of this year's conference is Mobile Graphics, that is, graphics on mobile devices, such as mobile phones, PDAs, and portable game consoles. As in previous years, we also welcome paper submissions in various other graphics areas.

The SIGRAD conference has an explicit ambition to broaden its geographic scope beyond the national borders of Sweden. We are therefore very happy to have several international contributions this year.

The keynote speakers this year are Michael Doggett from ATI, and Ulf Assarsson from Chalmers University of Technology. The topic of Michael's presentation is trends and recent developments in graphics hardware, and how that affects us all. Ulf presentation is about his work on soft shadow volumes in a broader perspective. Thanks so much for giving these presentations! We would also like to thank the program committee that provided timely reviews, and helped in selecting the papers for these proceedings.

Many thanks to our generous sponsors: Ericsson Mobile Platforms, TAT, and Aveva. We wish all participants a stimulating conference, and hope they take the chance and to create new connections in the Nordic graphics community.

Lennart Ohlsson
Program Chair

Tomas Akenine-Möller
Papers Chair

SIGRAD 2005

The SIGRAD 2005 program committee consisted of experts in the field of computer graphics and visualisation from Scandinavia and USA. We thank them for their comments and reviews.

Conference Chair

Lennart Ohlsson, Lund University

Program Chair

Tomas Akenine-Möller, Lund University

Program Committee Members

Timo Aila, Hybrid Graphics & Helsinki University of Technology, Finland

Ulf Assarsson, Chalmers University of Technology, Sweden

Niels Jørgen Christensen, Technical University of Denmark, Denmark

Anders Hast, University of Gävle, Sweden

Lars Kjelldahl, Royal Institute of Technology, Sweden

Ken Museth, Linköping University, Sweden

Petri Nordlund, Bitboys, Finland

Mikael Persson, TAT AB, Sweden

Kari Pulli, Nokia, USA

Stefan Seipel, University of Gävle, Sweden

Jacob Ström, Ericsson Research, Sweden

Edvard Sørgård, Falanx, Norway

Anders Ynnerman, Linköping University, Sweden

Conference Main Sponsor

Ericsson AB

Conference Sponsors

AVEVA AB

TAT AB

SIGRAD Board for 2005

Anders Backman, Chair

Stefan Seipel, Vice Chair

Lars Kjelldahl, Treasurer

Kai-Mikael Jää-Aro, Secretary

Anders Hast, Member

Odd Tullberg, Member

Anders Ynnerman, Member

Kenneth Bodin, Substitute

Thomas Larsson, Substitute

Ken Museth, Substitute

Lennart Ohlsson, Substitute

Örjan Vretblad, Substitute

Connected Minimal Acceleration Trigonometric Curves

Tony Barrera *
Barrera Kristiansen AB

Anders Hast †
Creative Media Lab,
University of Gävle

Ewert Bengtsson ‡
Centre for Image Analysis
Uppsala University

Abstract

We present a technique that can be used to obtain a series of connected minimal bending trigonometric splines that will intersect any number of predefined points in space. The minimal bending property is obtained by a least square minimization of the acceleration. Each curve segment between two consecutive points will be a trigonometric Hermite spline obtained from a Fourier series and its four first terms. The proposed method can be used for a number of points and predefined tangents. The tangent length will then be optimized to yield a minimal bending curve. We also show how both the tangent direction and length can be optimized to give as smooth curves as possible. It is also possible to obtain a closed loop of minimal bending curves. These types of curves can be useful tools for 3D modelling, etc.

Keywords: Trigonometric curves, Hermite curves, least square minimization

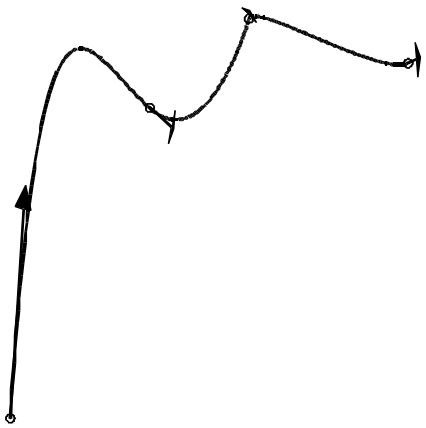


Figure 1: Multiple connected trigonometric curve with non optimal predefined tangents.

1 Introduction

This paper proposes a simple technique that will make it possible to construct a minimal bending curve through a number of consecutive points in space, using trigonometric splines [Schoenberg 1964]. Thus, each curve consists of a number of connected trigonometric Hermite spline segments [Alba-Fernandez 2004]. Each spline will start in one predefined point and end in the consecutive point, and the next curve segment will start in that point and end in the next point, and so forth.

In [Barrera 2005] a similar technique is presented where a minimal bending cubic curve is obtained where both the points and the directions at these points are given. That algorithm will compute optimal tangent lengths. Bartels et al [Bartels 1998] show how a minimal bending cubic curve can be obtained using the points only as constraints for the curve. The resulting splines will be Hermite splines and should not be confused with Catmul-Rom splines [Catmull 1974] which also intersect the given points. However, they are constructed in a quite different way.

Figure 1 shows a trigonometric Hermite curve where four points and tangents are set as constraints. The left part of the curve has rather large tangents, which make the curve bend heavily around the intersection points. On the right side the tangents are rather short, which makes the curve bend rapidly around the intersection points. A minimal bending curve will have minimal acceleration over the curve and this will make the curve smoother. Note that the length of the tangents have been scaled down to 25% in all the figures so that the tangents will not be too large compared to the curve.

Several such curves will be presented in this paper. First we will prove that a cubic curve that only has points as constraints will have this minimal acceleration property. The derivation will serve as an example when we proceed to discuss trigonometric curves instead. These curves have the advantage that they can define everything from straight lines to perfect circles. Next we will show how a trigonometric curve using points as constraints can be obtained and then we will show how a curve using both points and tangent directions can be constructed. In the latter case the tangent is set to an optimal, while in the first case both tangent length and direction is set to an optimal. Hence this type of curve will always be smoother, but we loose the possibility to determine direction in each point, which might be desirable for camera movements [Vlachos 2001] etc.

2 Least Square Minimization of Cubic Hermite Curves

We will start by proving that a cubic Hermite curve [Hearn 2004] that intersects a number of given points will actually have the minimal acceleration property. This will serve as an example of how the minimal acceleration is obtained since the equations are shorter and

*e-mail: tony.barrera@spray.se
†e-mail: aht@hig.se
‡e-mail: ewert@cb.uu.se

easier to understand than for trigonometric curves. Then we will go on to give examples of how this works for trigonometric curves.

The total curvature of a curve $f(t)$ in the parametric interval $[0,1]$ of one single curve segment is defined by

$$\int_0^1 \|\kappa(t)\|dt \tag{1}$$

Where $\kappa(t)$ is the curvature of the curve at t. This formula often causes very complex expressions, so it is more common to use the integral

$$\int_0^1 \|\mathbf{f}''(t)\|^2 dt \tag{2}$$

This integral sums the acceleration, i.e. the square of the second derivative over the curve. The acceleration is minimized by differentiating on some variable and set the result to zero so that the minimum is obtained. This is the essence of least square minimization [Burden 1989]. In our case we would like to find the optimal tangents that will give a minimal bending curve. If there are $k+1$ number of points, then there will be k number of curve segments. Hence we differentiate on the tangents and solve

$$\frac{\partial}{\partial \mathbf{T}_i}\int_0^1 \|\mathbf{f}_1''(t)\|^2 + \|\mathbf{f}_2''(t)\|^2 + ... + \|\mathbf{f}_{k+1}''(t)\|^2 dt = 0 \tag{3}$$

where $i = 1,2,...,k+1$.

In order to be able to derive the curve we must first compute the second derivatives of the Hermite curve. A general cubic curve is defined by

$$\mathbf{f}(t) = \mathbf{A}t^3 + \mathbf{B}t^2 + \mathbf{C}t + \mathbf{D} \tag{4}$$

and a Hermite curve has the initial conditions

$$\mathbf{f}(0) = \mathbf{P}_i \tag{5}$$
$$\mathbf{f}(1) = \mathbf{P}_{i+1} \tag{6}$$
$$\mathbf{f}'(0) = \mathbf{T}_i \tag{7}$$
$$\mathbf{f}'(1) = \mathbf{T}_{i+1} \tag{8}$$

Where $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$ are two tangent vectors to be determined for minimum acceleration. The Hermite [Hearn 2004] curve is defined by solving the system

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_i \\ \mathbf{P}_{i+1} \\ \mathbf{T}_i \\ \mathbf{T}_{i+1} \end{bmatrix} \tag{9}$$

The solution of this system is

$$\mathbf{A} = \mathbf{T}_i + \mathbf{T}_{i+1} - 2\mathbf{P}_{i,i+1} \tag{10}$$
$$\mathbf{B} = 3\mathbf{P}_{i,i+1} - 2\mathbf{T}_i - \mathbf{T}_{i+1} \tag{11}$$
$$\mathbf{C} = \mathbf{T}_i \tag{12}$$
$$\mathbf{D} = \mathbf{P}_i \tag{13}$$

where $\mathbf{P}_{i,i+1} = \mathbf{P}_{i+1} - \mathbf{P}_i$.

Hence

$$\mathbf{f}''(t) = 6\mathbf{A}t + 2\mathbf{B} \tag{14}$$

and

$$\|\mathbf{f}''(t)\|^2 = 36\mathbf{A}^2t^2 + 24\mathbf{A}\cdot\mathbf{B}t + 4\mathbf{B}^2 \tag{15}$$

where we use the notation $\mathbf{A}^2 = \mathbf{A}\cdot\mathbf{A}$ and so forth. Substituting equations (10) through (13) into equation (15) and differentiating on $\mathbf{T}_1$ as in equation (3) gives

$$\frac{\partial}{\partial \mathbf{T}_1}\int_0^1 \|\mathbf{f}''(t)\|^2 dt = 8\mathbf{T_1} + 4\mathbf{T_2} - 12\mathbf{P_{12}} \tag{16}$$

Moreover we have

$$\frac{\partial}{\partial \mathbf{T}_2}\int_0^1 \|\mathbf{f}''(t)\|^2 dt = 4\mathbf{T_1} + 16\mathbf{T_2} + 4\mathbf{T_3} - 12\mathbf{P_{12}} - 12\mathbf{P_{23}} \tag{17}$$
$$\frac{\partial}{\partial \mathbf{T}_3}\int_0^1 \|\mathbf{f}''(t)\|^2 dt = 4\mathbf{T_2} + 16\mathbf{T_3} + 4\mathbf{T_4} - 12\mathbf{P_{23}} - 12\mathbf{P_{34}} \tag{18}$$

and finally

$$\frac{\partial}{\partial \mathbf{T}_{k+1}}\int_0^1 \|\mathbf{f}''(t)\|^2 dt = 8\mathbf{T_k} + 4\mathbf{T_{k+1}} - 12\mathbf{P_{k,k+1}} \tag{19}$$

Next we set each equation equal to zero and solve for each tangent. After dividing each equation by four this yields a system of equations

$$\begin{bmatrix} 2 & 1 & 0 & 0 & ... & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & ... & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & ... & 0 & 0 & 0 \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ 0 & 0 & 0 & 0 & ... & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \\ \mathbf{T}_3 \\ . \\ . \\ . \\ \mathbf{T}_{k+1} \end{bmatrix} = \begin{bmatrix} 3\mathbf{P}_{12} \\ 3(\mathbf{P}_{12}+\mathbf{P}_{23}) \\ 3(\mathbf{P}_{23}+\mathbf{P}_{34}) \\ . \\ . \\ . \\ 3\mathbf{P}_{k,k+1} \end{bmatrix} \tag{20}$$

A system involving a matrix of this form is called a *tridiagonal system* and can be solved efficiently using a specialized algorithm [Lengyel 2004]. This is the same system, which is derived in [Bartels 1998]. However, they derive it in a different way were they set up a system requiring C^2 continuity at the intersection points. Nevertheless, our derivation proves that this type of curve have the minimal acceleration property.

3 Trigonometric Hermite splines

Trigonometric splines (or trigonometric polynomials) were introduced by Schoenberg [Schoenberg 1964] and have been investigated extensively in math and computer aided geometry literature, [Walz 1997], [Lyche 1979], [Han 2003], just to mention a few. However, they have not gained much interest in computer graphics. One reason is probably that it involves the computation of trigonometric functions and those have been computationally expensive. With faster hardware they may gain the interest from the computer graphics community as a modelling tool, since it is possible to construct everything from straight lines to perfect circle arcs. The latter is impossible with cubic curves.

A trigonometric spline can be constructed from a truncated Fourier series [Schoenberg 1964], [Walz 1997]. An Hermite spline is defined by two points and the tangents in these points and therefore we have four constraints and thus we need four terms in the Fourier series. The trigonometric curve is therefore defined as

$$\mathbf{f}(\theta) = \mathbf{a} + \mathbf{b}\cos\theta + \mathbf{c}\sin\theta + \mathbf{d}\cos 2\theta \tag{21}$$

Using the conditions in (5) through (8), the curve is found by solving

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix} \tag{22}$$

The solution is

$$\mathbf{a} = \frac{1}{2}(\mathbf{P}_1 + \mathbf{P}_2 - \mathbf{T}_1 + \mathbf{T}_2) \tag{23}$$

$$\mathbf{b} = -\mathbf{T}_2 \tag{24}$$

$$\mathbf{c} = \mathbf{T}_1 \tag{25}$$

$$\mathbf{d} = \frac{1}{2}(\mathbf{P}_1 - \mathbf{P}_2 + \mathbf{T}_1 + \mathbf{T}_2) \tag{26}$$

By forcing **d** to be equal to zero in equation (21) we get

$$\mathbf{f}(\theta) = \mathbf{a} + \mathbf{b}\cos\theta + \mathbf{c}\sin\theta \tag{27}$$

This is obviously the equation for a circle and this proves that it is possible to construct a perfect circle arc using these curves. Since the curve is parametric it is easy to see that it is possible to construct straight lines using the trigonometric splines. The coefficients are vectors and the function produces a point in space and each coordinate has its own expression and the only thing that differs is the coefficients, and therefore it is no problem to construct a straight line even though trigonometric functions are involved.

4 Least Square Minimization of Trigonometric Hermite Splines

Once again we use equation (3) in order to optimize both tangent length and direction for the Trigonometric Hermite spline defined in equation (21). This will yield a system of equations that must be solved.

$$\begin{bmatrix} A & 2B & 0 & 0 & ... & 0 & 0 & 0 \\ B & A & B & 0 & ... & 0 & 0 & 0 \\ 0 & B & A & B & ... & 0 & 0 & 0 \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ 0 & 0 & 0 & 0 & ... & 0 & 2B & A \end{bmatrix} \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \\ \mathbf{T}_3 \\ . \\ . \\ . \\ \mathbf{T}_{k+1} \end{bmatrix} = \begin{bmatrix} 2C\mathbf{P}_{12} \\ C(\mathbf{P}_{12} + \mathbf{P}_{23}) \\ C(\mathbf{P}_{23} + \mathbf{P}_{34}) \\ . \\ . \\ . \\ 2C\mathbf{P}_{k,k+1} \end{bmatrix} \tag{28}$$

where we have

$$A = 15\pi - 16 \tag{29}$$

$$B = 6\pi - 11 \tag{30}$$

$$C = 6\pi - 4 \tag{31}$$

Figure 2 shows how the proposed approach will yield a curve that is much smoother than the curve in figure 1, since both the tangent direction and length are set to an optimal, giving a minimal bending curve.

4.1 Optimal tangent length

If we want our curves to have the same direction as the tangents in the intersection points, then we can change the computation so that we solve for optimal tangent length only instead of solving for both optimal tangent length and direction. In this case we introduce α_i as the length of each tangent $\mathbf{T_i}$.

The equation now becomes

$$\frac{\partial}{\partial \alpha_i} \int_0^1 \|\mathbf{f}_1''(t)\|^2 + \|\mathbf{f}_2''(t)\|^2 + ... + \|\mathbf{f}_{k+1}''(t)\|^2 dt = 0 \tag{32}$$

where $i = 1, 2, ..., k+1$.

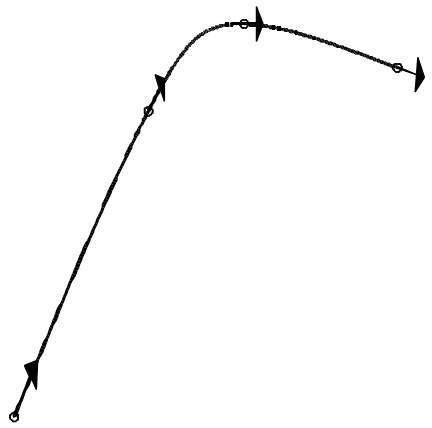


Figure 2: Multiple connected minimal acceleration trigonometric curves, with both optimal tangent direction and length.

The resulting coefficient matrix is

$$\begin{bmatrix} A\mathbf{T}_1^2 & 2B\mathbf{T}_1 \cdot \mathbf{T}_2 & 0 & 0 & ... & 0 & 0 & 0 \\ B\mathbf{T}_1 \cdot \mathbf{T}_3 & A\mathbf{T}_2^2 & B\mathbf{T}_2 \cdot \mathbf{T}_3 & 0 & ... & 0 & 0 & 0 \\ 0 & \mathbf{T}_2 \cdot \mathbf{T}_3 & A\mathbf{T}_3^2 & B\mathbf{T}_3 \cdot \mathbf{T}_4 & ... & 0 & 0 & 0 \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ 0 & 0 & 0 & 0 & ... & 0 & 2B\mathbf{T}_k \cdot \mathbf{T}_{k+1} & A\mathbf{T}_{k+1}^2 \end{bmatrix} \tag{33}$$

And the variables to solve for are

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ . \\ . \\ . \\ \alpha_{k+1} \end{bmatrix} \tag{34}$$

Finally the column of constants is

$$\begin{bmatrix} 2C\mathbf{T}_1 \cdot \mathbf{P}_{12} \\ C\mathbf{T}_2 \cdot (\mathbf{P}_{12} + \mathbf{P}_{23}) \\ C\mathbf{T}_3 \cdot (\mathbf{P}_{23} + \mathbf{P}_{34}) \\ . \\ . \\ . \\ 2C\mathbf{t}_{k+1} \cdot \mathbf{P}_{k,k+1} \end{bmatrix} \tag{35}$$

In figure 3 it is clear that the tangents have the same directions as in figure 1. However, the tangents have optimal length and the curve is thus smoother.

5 A Closed Loop

It is possible to connect any number of minimal acceleration trigonometric curves together into a closed loop as shown in figure 4. The end point for the last segment is set to be the same as the start point for the first segment. Likewise, the tangents at this

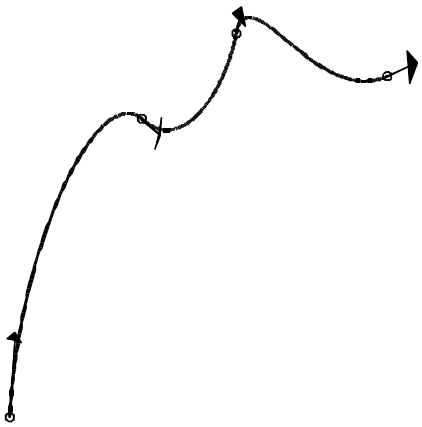


Figure 3: Multiple connected minimal acceleration trigonometric curve.

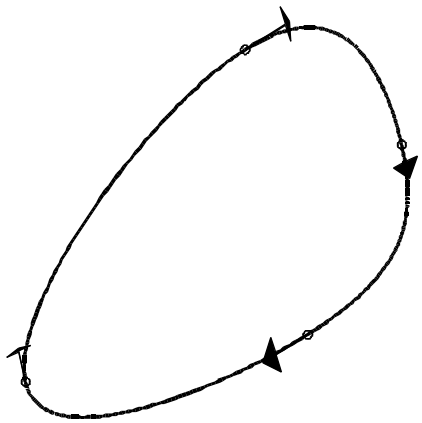


Figure 4: A closed loop of a trigonometric curve with optimal tangent length and direction

point is set in the same way. The equation to solve is now changed so that we have

$$\frac{\partial}{\partial \mathbf{T}_i}\int_0^1 \|\mathbf{f}_1''(t)\|^2 + \|\mathbf{f}_2''(t)\|^2 + ... + \|\mathbf{f}_k''(t)\|^2 dt = 0 \qquad (36)$$

where $i = 1,2,...,k$. Note that this time there are k number of points and k number of curve segments.

This yields the following system

$$\begin{bmatrix} A & B & 0 & 0 & ... & 0 & 0 & B \\ B & A & B & 0 & ... & 0 & 0 & 0 \\ 0 & B & A & B & ... & 0 & 0 & 0 \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ B & 0 & 0 & 0 & ... & 0 & B & A \end{bmatrix} \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \\ \mathbf{T}_3 \\ . \\ . \\ . \\ \mathbf{T}_k \end{bmatrix} = \begin{bmatrix} C(\mathbf{P}_{k,1}+\mathbf{P}_{12}) \\ C(\mathbf{P}_{12}+\mathbf{P}_{23}) \\ C(\mathbf{P}_{23}+\mathbf{P}_{34}) \\ . \\ . \\ . \\ C(\mathbf{P}_{k-1,k}+\mathbf{P}_{k,1}) \end{bmatrix} \qquad (37)$$

where we have

$$A = 15\pi - 16 \qquad (38)$$

$$B = 6\pi - 11 \qquad (39)$$

$$C = 6\pi - 4 \qquad (40)$$

The presence of the nonzero entries in the lower-left and upper-right corners make this system a *cyclic tridiagonal system*. It can also be solved efficiently [Press 1992].

5.1 Optimal Tangent Length

Now we proceed to show how a closed loop can be constructed when we want a specific tangent direction in each point. The equation to solve is

$$\frac{\partial}{\partial \alpha_i}\int_0^1 \|\mathbf{f}_1''(t)\|^2 + \|\mathbf{f}_2''(t)\|^2 + ... + \|\mathbf{f}_k''(t)\|^2 dt = 0 \qquad (41)$$

where $i = 1,2,...,k$.

The coefficient matrix now becomes

$$\begin{bmatrix} A\mathbf{T}_1^2 & B\mathbf{T}_1\cdot\mathbf{T}_2 & 0 & 0 & ... & 0 & 0 & B\mathbf{T}_1\cdot\mathbf{t}_k \\ B\mathbf{T}_1\cdot\mathbf{T}_3 & A\mathbf{T}_2^2 & B\mathbf{T}_2\cdot\mathbf{T}_3 & 0 & ... & 0 & 0 & 0 \\ 0 & \mathbf{T}_2\cdot\mathbf{T}_3 & A\mathbf{T}_3^2 & B\mathbf{T}_3\cdot\mathbf{T}_4 & ... & 0 & 0 & 0 \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ B\mathbf{T}_1\cdot\mathbf{T}_k & 0 & 0 & 0 & ... & 0 & 2B\mathbf{T}_{k-1}\cdot\mathbf{T}_k & A\mathbf{T}_k^2 \end{bmatrix} \qquad (42)$$

And the variables to solve for are

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ . \\ . \\ . \\ \alpha_k \end{bmatrix} \qquad (43)$$

Finally the column of constants is

$$\begin{bmatrix} C\mathbf{T}_1\cdot(\mathbf{P}_{12}+\mathbf{P}_{k,1}) \\ C\mathbf{T}_2\cdot(\mathbf{P}_{12}+\mathbf{P}_{23}) \\ C\mathbf{T}_3\cdot(\mathbf{P}_{23}+\mathbf{P}_{34}) \\ . \\ . \\ . \\ C\mathbf{T}_k\cdot(\mathbf{P}_{k-1,k}+\mathbf{P}_{k,1}) \end{bmatrix} \qquad (44)$$

In figure 5 the tangent directions have been predefined. The curve is made smooth by the proposed minimal acceleration technique, so that the tangent length is set to an optimal.

6 Conclusions

We have presented a method that can be used to obtain minimal bending trigonometric Hermite curves, which can have a number of different constraints, like intersection points and tangent directions. These curves can be used in a number of areas, such as 3D-modeling and camera movements.

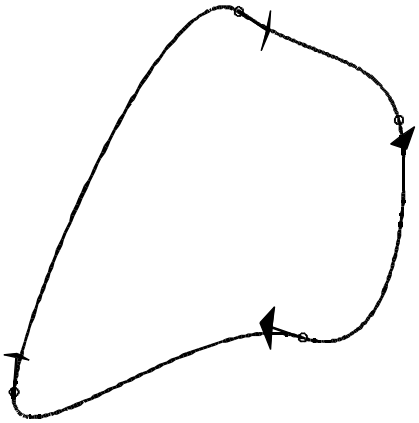


Figure 5: A closed loop with optimal tangent length for predefined tangent directions.

References

V. ALBA-FERNANDEZ, M.J. IBANEZ-PEREZ, M.D. JIMENEZ-GAMERO 2004. *A Bootstrap Algorithm for the Two-Sample Problem Using Trigonometric Hermite Spline Interpolation* Communications in Nonlinear Science and Numerical Simulation. Vol. 9. Num. 2. Pag. 275-286

T. BARRERA, A. HAST, E. BENGTSSON 2005. *Minimal Acceleration Hermite Curves* Graphics Programming Gems V, Charles River Media, Edited by Kim Pallister, pp 225-231.

R.H. BARTELS, J.C. BEATTY,AND B.A. BARSKY 1998. *An Introduction to. Splines for use in Computer Graphics and. Geometric Modeling* "Hermite and Cubic Spline Interpolation." Ch. 3, pp. 9-17.

R. L. BURDEN, J.D. FAIRES 1989. *Numerical Analysis* Numerical Analysis, PWS-KENT Publishing company Boston, pp. 439, 440.

E. CATMULL, R. ROM 1974. *A Class of Local Interpolating Splines* Computer Aided Geometric design, pp. 317-326.

A. FOLEY, J. D., ET AL 1997. *Computer Graphics: Principles and Practice, 2nd ed.* Addison-Wesley, p. 480.

X. HAN 2003. *Piecewise Quadratic Trigonometric Polynomial Curves* Mathematics of Computation, pp. 1369-1377.

D. HEARN, M.P BAKER 2004. *Computer Graphics with OpenGL* Pearson Education Inc., pp. 426-429.

E. LENGYEL. 2004. *E. Lengyel, Mathematics for 3D Game Programming and Computer Graphics, 2nd ed.* E. Lengyel, Charles River Media, pp. 433-436.

T. LYCHE. 1979. *A Newton form for Trigonometric Hermite Interpolation* E. Lengyel, Charles River Media, pp. 433-436.

PRESS ET AL. 1992. *Numerical Recipes in C.* [Press92] Cambridge University Press, pp. 74-75.

I. J. SCHOENBERG 1964. *On Trigonometric Spline Interpolation* Journal of Mathematics and Mechanics, pp. 795-825.

A. VLACHOS, J ISIDORO *Smooth C^2 Quaternion-based Flythrough Paths* Game Programming Gems 2, Charles River Media, Edited by Mark Deloura, pp. 220-227.

G. WALZ *Identities for Trigonometric B-Splines with an Application to Curve Design* Identities for trigonometric B-splines with an application to curve design. BIT 37, pp. 189-201.

Texture Compression: THUMB – Two Hues Using Modified Brightness

Martin Pettersson Jacob Ström

Ericsson Research

Abstract

We present a new texture compression system called THUMB, that can be used either as a stand-alone compression system or in combination with the *i*PACKMAN algorithm. We show how the combined system improves quality in the test images we have used, especially in the image blocks most problematic to *i*PACKMAN.

1 Introduction

Bandwidth is usually the factor limiting performance in rasterization-based rendering hardware [Aila et al. 2003]. Knittel et al. [1996] and Beers et al [1996] show how *texture compression* can be used to reduce bandwidth during rendering. By transferring the texels over the bus in compressed form, and decompressing needed texels on-the-fly, texture bandwidth can be reduced significantly. Previously introduced image compression techniques such as the CCC scheme [Campbell et al. 1986] can be used for the compression.

For mobile devices, which are powered by batteries, these bandwidth savings are also important from a power consumption perspective, since such off-chip memory accesses are often the most energy consuming operations in a computer system [Fromm et al. 1997]. In low-power processes, such as the ones used for mobile devices, off-chip memory accesses are more than an order of magnitude more energy consuming than accesses to a small on-chip SRAM memory. The bandwidth savings can therefore be translated into energy savings. The texture compression system presented here was originally intended for use on mobile devices, but could be used on PC systems and game consoles as well.

A texture compression system differs from a normal image compression system in a number of ways. First, it needs to allow random access to the texels, since rendering can start in any location in the texture, and be traversed in a non-scanline fashion. Most texture compression systems are therefore block-based fixed rate codecs, where each block of the image is given a fixed number of bits, which makes it simple to calculate the address of a particular block. Second, the decompression of a block should ideally be of low complexity. If some sort of filtering is used, many parallel decompression units are needed to process a single pixel. For instance, if trilinear filtering is used, eight parallel decompression units are needed to process one pixel per clock. By decompressing the texels right before filtering, it is possible to keep compressed texels in the texture cache, which means that the cache can be made several times smaller in terms of chip surface area. Third, it is advantagous to avoid texture dependent look-up tables (LUTs), such as color palettes, since the indirect addressing they introduce give rise to latencies that are hard and costly to hide.

Our new texture compression system is developed with *i*PACKMAN texture compression [Ström and Akenine-Möller 2005] in mind, and designed to be a complementing mode in that coder, taking care of the blocks that *i*PACKMAN has most difficulties with. However, it can also be used as a stand-alone codec, and we have presented results for both usages.

2 Previous Work

We will now go through previous work that is related to texture compression.

Delp and Michell [1979] present a scheme called block truncation coding (BTC) for gray scale images. The image is divided into 4×4 blocks, and two shades of gray are encoded in the block, together with a bit mask that decides for each pixel what shade to choose. The bit mask is thus 16 bits, and eight bits are used for each gray level, resulting in 32 bits per 4x4 block or 2 bits per pixel (bpp).

Campbell et al [1986] extend the BTC algorithm to color images in a system called CCC — Color Cell Compression. Each 4×4 block now includes two colors instead of two gray scales. By using a 256-wide color palette, the colors can be represented with eight bits each, yielding 2 bpp for color images. However, only two colors are possible per block, which limits image quality. Furthermore, having a color palette is a drawback in today's systems, where memory accesses are slow in relation to computation.

The de facto standard today is the S3TC texture compression method by Iourcha et al. [1999], and it can be seen as an extension of CCC. To increase quality compared to CCC, four colors can be chosen in each pixel, yielding two bits per pixel in the bit mask. To avoid a texture dependent LUT, no color palette is used. Instead two colors in RGB565 format are stored in the block, and two more colors are interpolated in-between these two colors. This means that colors in a block are restricted to lying on a line in RGB space. However, this is a rather good approximation of the color distribution in blocks from most natural images. An example can be seen in the left diagram in Figure 1, which shows a cross section of the RGB space, and where the colors of a block are plotted as points in RGB space. The point cloud is approximated by four equidistant reconstruction points along a line, as shown in the right diagram. The two end points (marked with squares) are the colors stored in the block, whereas the two middle points (marked with circles) are interpolated. With 64 bits per 4×4 block, the rate of S3TC is 4 bpp.

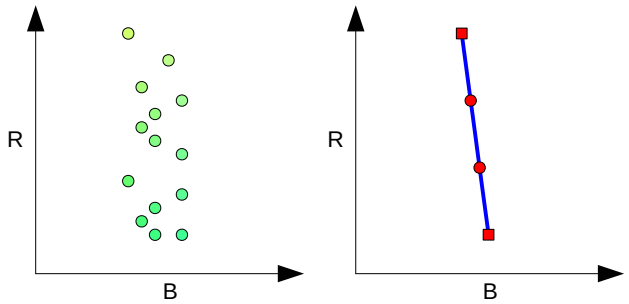


Figure 1: Left: Possible distribution of the colors of a 4×4 block in RGB-space. Right: In S3TC the colors are approximated by four equidistant points along a line.

Akenine-Möller and Ström [2003] present a variant of S3TC called POOMA, where the biggest difference is that only one in-between color is used, and blocks are 2×3 pixels. Here the main

Figure 2: Here, the core idea of PACKMAN is illustrated. To the left, the base color for each 2×4 block is shown. The image in the middle shows the per pixel luminance modulation. The rightmost image shows the decompressed image.

target is to reach 32 bits per block to match bus-sizes of mobile phones on a rendering system without a texture cache. However, this reduces quality and increases the rate as compared to S3TC, and the block size of 2×3 pixels is awkward for hardware implementation.

Beers et al. [1996] use vector quantization for texture compression to reach compression rates of 1-2 bpp. However, this requires a big LUT, which introduces indirect addressing resulting in latencies that can be hard to hide.

Fenney [2003] uses a different approach, exploiting the fact that an upscaled low-resolution version of an image is often similar to the image itself. Fenney uses two such low-resolution images A and B, both upscaled bilinearly two times, yielding only one color sample each per 4×4 block. Each pixel can then choose its color from either image A, image B, or from two blend values between A and B. Using 16 bits for each color and 2 bits per pixels for choosing the blend value results in 64 bits per block or 4 bpp. A 2 bpp mode is also present.

Since our work is built on the PACKMAN [Ström and Akenine-Möller 2004] and *i*PACKMAN algorithms, we will go through them in more detail in the next section.

3 PACKMAN and iPACKMAN

The PACKMAN algorithm exploits the fact that the human visual system is more sensitive to changes in luminance than in chrominance. It takes the rather radical approach of only having a single chrominance per 2×4 pixel block, represented as a RGB444 color (12 bits). Each pixel can then modify the luminance of this base color additively, as shown in Figure 2. More specifically, a *modifier value* is added to all three components (R, G and B) of the base color. The modifier value is taken from a small table of four entries, and hence two bits, called *pixel indices* are needed to select the value for each pixel. Finally, four bits are spent on a *table codeword*, to select the small table from a list of 16 prefixed tables. Altogether, 32 bits are used for 2×4 pixels, giving a rate of 4 bpp.

3.1 iPACKMAN

This algorithm has been improved under the name *i*PACKMAN (also called Ericsson Texture Compression, ETC) in two ways [Ström and Akenine-Möller 2005]. First and most important, a differential mode is introduced, allowing two neighboring 2×4 blocks to be coded together. The base color of the left block can then be encoded using RGB555, i.e., with higher precision, and the right base color also in RGB555 format, but coded using a differential dRdGdB333, where dR, dG and dB can assume values between -4 and $+3$. Thus, for pairs of blocks with similar base colors, the chrominance resolution effectively goes up from RGB444 to RGB555 in both blocks. Blocks that cannot be encoded well using the differential mode will be coded as before, i.e., with two individually coded RGB444 colors. This mode is called the individual mode.

Figure 3: Left: Original. Right: Image compressed using *i*PACKMAN. Note the blocky artifacts coming from that only one hue is allowed per subblock (not visible in b/w reproduction).

The second improvement is that blocks can be flipped so that a 4×4 block consists of either two 2×4 block next to each other, or two 4×2 blocks on top of each other. Two mode bits are needed, one to choose between individual and differential mode, and one to indicate the flip status. Space for these two bits are created by shrinking the number of possible tables from 16 to eight, thus reducing the number of table bits in each sub-block from four to three. Figure 8 shows the bit layout in the differential (top) and the individual (bottom) modes.

These two small differences have a substantial effect on image quality, which jumps 2.5 dB in terms of Peak Signal to Noise Ratio (PSNR), suddenly putting *i*PACKMAN on par with S3TC. Visually, *i*PACKMAN lacks the disturbing banding artifacts that are a result of the low chrominance resolution in PACKMAN. However, *i*PACKMAN does not change the fact that only one chrominance can be used for a block of eight pixels.

4 THUMB Texture Compression

In this section, we present our new THUMB texture compression scheme. First we describe the design and motivate the different design choices. Then follows descriptions for decompression and compression of the stand-alone version of THUMB. The last subsection describes how THUMB can be combined with *i*PACKMAN to get a better solution.

4.1 Basic Design and Motivation

Almost all fixed-rate block compression techniques have particular blocks that are coded worse than others. This is also the case for *i*PACKMAN. The overall performance of *i*PACKMAN is very good, but subblocks with more than one distinct hue are sometimes coded with poor result as the pixel value can only be modified in the direction $(1, 1, 1)$ in RGB space. An image with typical problem blocks is shown in Figure 3. The image is cropped from a larger image showing a road with a yellow line. Since the human visual system is good at picking up block artifacts such as these, the main goal of this paper is to be able to better handle such blocks. However, we do not want to compromise quality in other blocks in order to reach that goal, so our secondary goal is that overall quality should stay the same or increase.

Our new scheme is called *Two Hues Using Modified Brightness*, or THUMB for short. Just like *i*PACKMAN, it is based on 4×4 blocks. In the stand-alone version, each block is coded with 64 bits. As the name suggests, THUMB can handle up to two different hues

in a block. To allow this, two independent base colors are used to code each block. This is also the case for *i*PACKMAN where each base color is restricted to a 2×4 subblock. In THUMB however, each base color can be used for any pixel in a block. The colors are encoded in RGB554 format.

Modifying the brightness for each pixel has proved to be a good solution for *i*PACKMAN. Therefore this approach has been used in THUMB as well. The technique however is somewhat different. 32 bits are used as pixel indices, giving us two bits to code each one of the 16 pixels in a block. Hence, we can have four different paint colors.

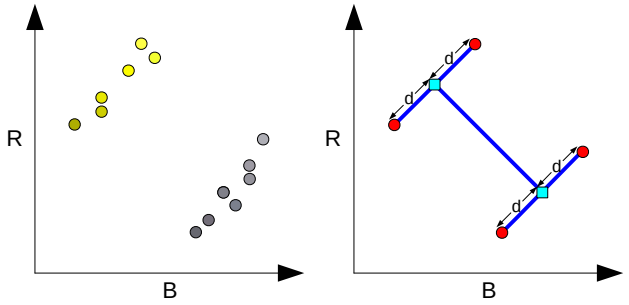


Figure 4: Left: The colors of the original block can be located in two different hues in RGB space. Right: Two base colors (marked with squares) are selected. Four paint colors (marked with circles) are derived by adding a distance d in direction $(1,1,1)$ to form an H-pattern.

These paint colors can be chosen in several ways since we are using two independent base colors per block. THUMB defines two different patterns of how to retrieve the four paint colors. In the first pattern, these paint colors are derived using a distance d added in direction $(1,1,1)$ from the base colors. This pattern is called the H-pattern, since the paint colors and the base colors can be placed as an H in RGB space. The H-pattern is illustrated in Figure 4, where a cross section of the RGB space is shown. Note that, just as in S3TC, the colors are approximated with line segments. Whereas S3TC can choose any orientation of the line segment (see Figure 1), the line segments in THUMB must be oriented parallel with the intensity direction $(1,1,1)$. On the other hand, THUMB can use two line segments whereas S3TC can use only one.

Sometimes the colors of the original block are clustered more around one base color than the other. A different pattern might then be a better match. In the second pattern, the first two paint colors are the base colors themselves. To get the other two paint colors, the distance d is added to the first base color in direction $(1,1,1)$. In this way, three paint colors with the same hue can be represented in a block. The fourth paint color is then chosen independently from the first three. This pattern is called the T-pattern, since the paint colors can be placed to form a T in RGB space as illustrated in Figure 5. A *pattern bit* is used to resolve which one of the two patterns to use when generating the paint colors. The patterns are approximately equally common. The distance d is coded using three bits in the stand-alone version. As a total we will have eight possible distance values to choose from for each block. The distances are taken from a hardware lookup table. This table was created using a combination of different optimizing techniques for a test suite of twenty images. The optimized table is shown in Table 1. We have tried approximating the tables using shifts as well, and even though it only gives a small performance loss, it is not clear that we actually would gain much in terms of HW complexity.

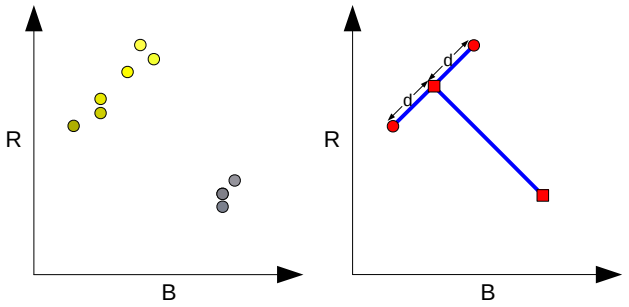


Figure 5: Left: An uneven distribution of the original block colors. Right: The T-pattern. Both base colors are used as paint colors. The distance d is added in direction $(1,1,1)$ to get the other two.

table index	0	1	2	3	4	5	6	7
distance	3	6	11	16	23	32	41	64

Table 1: Hardware lookup table with optimized distances.

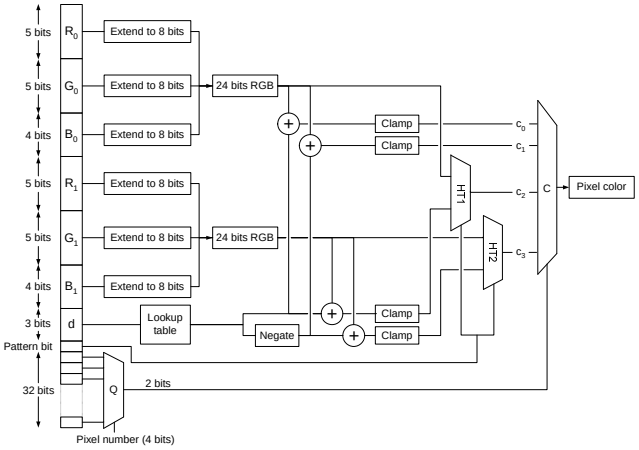


Figure 6: Hardware diagram for a possible THUMB decoder. The bit layout can be seen to the left. The expanded color components have been combined to RGB-colors to make the diagram more readable.

4.2 Decompression

Figure 6 shows a possible implementation of how to decode a pixel in THUMB. At most, the values of twenty bits are needed to retrieve a pixel. The two bits from the pixel indices are used to determine which one of the paint colors to decode. Below is a description of how each paint color is derived:

1. To retrieve the first paint color, the first base color must be read. Each component is expanded to eight bits to form a 24-bit RGB-color. The distance is read from the lookup table, and added to each component of the base color. The color components are then clamped to the interval $[0,255]$, resulting in the first paint color.

2. The second paint color is decoded in a similar way as the first one. The only difference is that the distance is negated before it is added to the components of the base color.

3. Decoding the third paint color is a bit trickier than the first two ones. Depending on the value of the *pattern bit*, two different

paint colors can be acquired. If the *pattern bit* is zero, the first base color is expanded and used as the paint color without adding a distance. If the *pattern bit* is set, the second base color is expanded and the distance is added to get the third paint color.

4. The *pattern bit* is also needed to determine the fourth paint color. If the *pattern bit* is zero, the second base color is expanded and used as paint color. If the *pattern bit* is set, the negated distance is added to all components of the expanded second base color to get the fourth paint color.

4.3 Compression

The problem of compression is to find the best possible pair of base colors. Exhaustive search is not yet feasible due to the number of combinations that must be tested. Iterating over all possible base colors (2^{28}), patterns (2), distances (2^3) and paint colors (4) means that up to 2^{34} different combinations would have to be tried for each pixel. Therefore, three non-exhaustive compression methods have been developed.

LBG Compression. The LBG vector quantization algorithm [Linde et al. 1980] is used to find the two base colors. Since we only have two reconstruction values (the base colors), the algorithm converges quite fast. Starting with two random base colors, only ten iterations are needed to get a satisfying result. After the base colors are found, all possible patterns, distances and paint colors are tried. The parameter combination giving the lowest Mean Square Error (MSE) is chosen for the block. Encoding a 512×512 texture takes less than five seconds using a 800 MHz PC with 256 MB of RAM.

Radius Compression. This method is much slower than LBG compression, but will also give a better result. Initially, two base colors are found using the LBG-algorithm as above. Then for each quantized base color, all possible colors within a $(2k+1) \times (2k+1) \times (2k+1)$ cube centered around the base color are tried. Loosely speaking, k can be called the radius of the cube, hence the name. The encoding time increases very quickly with respect to k, while the gain in image quality is decreasing: Since there are two colors per block, and they cannot be tested independently, radius compression is $(2k+1)^6$ times slower than LBG compression. For instance, radius compression with $k = 1$ is 729 times slower than LBG compression, $k = 2$ is 15625 times slower and so on. In practice, the extra encoding time for a radius level over two will not justify the small gain in quality. The gain in image quality using the first level of radius is on average around 1 dB in terms of Peak Signal to Noise Ratio (PSNR), compared to LBG compression.

Selective Compression. This solution exploits the fact that all surrounding colors in radius compression are not equally probable. Empirical studies show that for almost all blocks where the first radius level is used, it is sufficient to try only the colors shown in figure 7. This means that only nine different colors need to be tested for each base color instead of 27. Thus, the encoding time is decreased a factor nine, compared to radius compression. The loss in image quality however, is only about 0.05 dB.

4.4 Combining THUMB and iPACKMAN

Blocks containing two distinct hues are coded very well with THUMB. However, the overall performance is in general slightly worse than for *i*PACKMAN. This is mainly because *i*PACKMAN

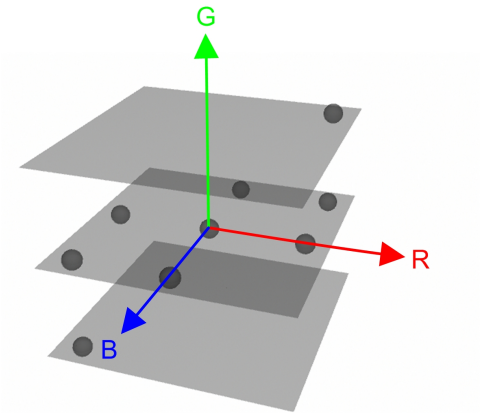


Figure 7: All colors tried in radius search are not equally common. In selective compression, only the most frequent colors from radius compression are tested. The figure shows the constellation of these colors for the first level of radius.

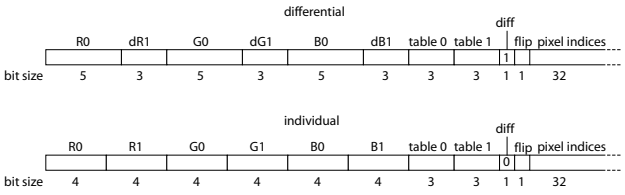


Figure 8: Modes in original *i*PACKMAN. Top: Differential mode. Bottom: Individual mode.

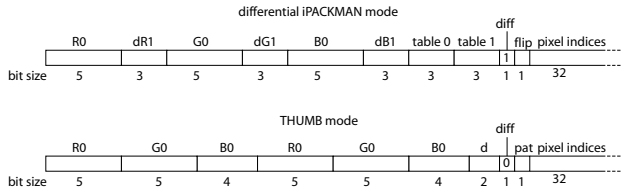


Figure 9: Modes in the combined *i*PACKMAN and THUMB coder. Note how the individual mode in *i*PACKMAN has been exchanged with a 63-bit THUMB mode.

has the possibility to have eight different paint colors per 4×4 block, as the blocks are divided into two subblocks.

Since *i*PACKMAN and THUMB are good at different types of blocks, it makes sense to combine the two. Each block is encoded using both *i*PACKMAN and THUMB separately. The one giving the lowest MSE will be used to represent the block. In order to fit both *i*PACKMAN and THUMB into 64 bits, both algorithms need to be represented using fewer bits.

How to do this is not obvious—the design space is truly huge—and we have only been able to cover a small number of the possible constellations. However, the solution that has given the best result among the ones we have tried is also the one that we think is the most straight-forward: The individual mode in *i*PACKMAN is replaced with a 63-bit THUMB mode, as can be seen in Figures 8 and 9. This has a number of advantages: Firstly, all decoding hardware for the individual mode in *i*PACKMAN can be removed and replaced by the THUMB mode. Also, when encoding, only two modes (differential, THUMB) need to be tested, just as in *i*PACKMAN (differential, individual). It is also likely that blocks

	Kodak img 1	Kodak img 2	Kodak img 3	Kodak img 4	Kodak img 5	Lena	Lorikeet	Avg gain
PVR-TC	33.8 [8.98]	37.1 [6.20]	37.9 [5.61]	37.7 [5.76]	32.4[10.59]	35.9 [7.11]	34.8 [8.08]	+0.85 dB
S3TC	34.78	36.86	38.53	37.96	32.80	35.97	34.37	+0.61 dB
*i*PACKMAN	36.29	38.09	38.62	38.59	34.12	35.17	33.25	+0.21 dB
Stand-alone THUMB	34.80	36.86	37.88	37.34	32.97	35.31	33.69	+0.96 dB
Combined Solution	36.26	38.12	38.92	38.66	34.08	35.66	33.88	—

Table 2: The PSNR is reported from a test suite of images for PVR-TC, S3TC, *i*PACKMAN, stand-alone THUMB and the combination of THUMB and *i*PACKMAN. The rightmost column shows the average gain when comparing the combined system to the other schemes.

that are coded using the individual mode in *i*PACKMAN will be well represented with THUMB, since two very different base colors are easily representable in THUMB.

Combining *i*PACKMAN with THUMB in this way means that THUMB must operate at 63 bits, since the diff bit occupies one bit. This is solved by using two instead of three bits for the distance table. The new optimized table contains the distances { 6 13 24 43 }.

5 Results

In this section we compare the image quality of different texture compression schemes. Our new schemes THUMB and the combined solution are compared to *i*PACKMAN, S3TC and PVR-TC.

To maximize image quality, the slowest compression mode of *i*PACKMAN has been used.

THUMB is encoded with radius compression using the second level of radius. This is also the case for the THUMB-mode in the combined solution.

S3TC is encoded using the Compressonator software package from ATI. The DirectX mode was used in this comparison, setting the weights to $(1,1,1)$ for the lowest error score.

There is no publicly available codec for PVR-TC, so the results are taken from Fenneys publication.

The results of PVR-TC was presented in *root mean squared error* (RMSE):

$$\sqrt{\frac{1}{w \times h}\sum_{x,y}(\Delta R_{xy}^2 + \Delta G_{xy}^2 + \Delta B_{xy}^2)}$$

where w and h are the width and the height of the image, and ΔR_{xy}, ΔG_{xy} and ΔB_{xy} are the pixel differences in pixel (x,y) between the original and the decompressed image in the red, green and blue component respectively. We have chosen to present our results in *Peak Signal to Noise Ratio* (PSNR) instead:

$$PSNR = 10\log_{10}\left(\frac{3\times 255^2}{RMSE^2}\right), \qquad (1)$$

where the scale factor 3 in the numerator is due to the fact that 3×255^2 is the peak energy in a pixel.

To be able to compare the results with PVR-TC, the same seven images used for the testing of PVR-TC have been used here. Just as in Fenney's study, these have been cropped to 512×512 pixels.

The results of the comparison between the different schemes are found in Table 2. It can be seen that the combined solution outperforms both S3TC and PVR-TC with over 0.5 dB. Using THUMB separately is almost one dB worse than using the combined solution. The same number for *i*PACKMAN is only 0.21 dB. Generally in image coding, a difference below 0.25 dB is hard to notice. However, the main goal of this paper was not to increase overall quality, but to handle the particular blocks that are most problematic for *i*PACKMAN, and this objective has been met. Examples can be seen in Figure 10. The secondary goal, that overall quality on average should stay the same or increase, has also been reached.

It must be said that seven images are not enough to make a good statistical analysis of the results. A larger test suite with images intended for texture mapping would be desirable. What can be said however, is that THUMB solves some of the worst problem blocks for *i*PACKMAN.

6 Conclusion

We have presented a new texture compression algorithm, THUMB, that can be used as a stand-alone system or in combination with *i*PACKMAN. If combined, THUMB can improve some of the worst blocks in *i*PACKMAN, and at the same time raise overall quality some. Still, some blocks, such as gradients between two colors illustrated by the explosion in Figure 10, are better handled by S3TC than with the proposed combination. This opens up for future work, and perhaps new modes.

References

AILA, T., MIETTINEN, V., AND NORDLUND, P. 2003. Delay Streams for Graphics Hardware. *ACM Transactions on Graphics, 22*, 3, 792–800.

AKENINE-MÖLLER, T., AND STRÖM, J. 2003. Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. *ACM Transactions on Graphics, 22*, 3, 801–808.

BEERS, A., AGRAWALA, M., AND CHADDA, N. 1996. Rendering from Compressed Textures. In *Proceedings of SIGGRAPH*, 373–378.

CAMPBELL, G., DEFANTI, T. A., FREDERIKSEN, J., JOYCE, S. A., LESKE, L. A., LINDBERG, J. A., AND SANDIN, D. J. 1986. Two Bit/Pixel Full Color Encoding. In *Proceedings of SIGGRAPH*, vol. 22, 215–223.

DELP, E., AND MITCHELL, O. 1979. Image Compression using Block Truncation Coding. *IEEE Transactions on Communications 2*, 9, 1335–1342.

FENNEY, S. 2003. Texture Compression using Low-Frequency Signal Modulation. In *Graphics Hardware*, ACM Press, 84–91.

FROMM, R., PERISSAKIS, S., CARDWELL, N., KOZYRAKIS, C., MCCAUGHY, B., PATTERSON, D., ANDERSON, T., AND YELICK, K. 1997. The Energy Efficiency of IRAM Architectures. In *24th Annual International Symposium on Computer Arhchitecture*, ACM/IEEE, 327–337.

IOURCHA, K., NAYAK, K., AND HONG, Z. 1999. System and Method for Fixed-Rate Block-based Image Compression with Inferred Pixels Values. In *US Patent 5,956,431*.

KNITTEL, G., SCHILLING, A., KUGLER, A., AND STRASSER, W. 1996. Hardware for Superior Texture Performance. *Computers & Graphics 20*, 4 (July), 475–481.

LINDE, Y., BUZO, A., AND GRAY., R. M. 1980. An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communication 28*, 1, 84–95.

STRÖM, J., AND AKENINE-MÖLLER, T. 2004. PACKMAN: Texture Compression for Mobile Phones. In *Sketches program at SIGGRAPH*.

STRÖM, J., AND AKENINE-MÖLLER, T. 2005. iPACKMAN: High Quality, Low Complexity Texture Compression for Mobile Phones. In *Graphics Hardware*, ACM Press, 63–70.

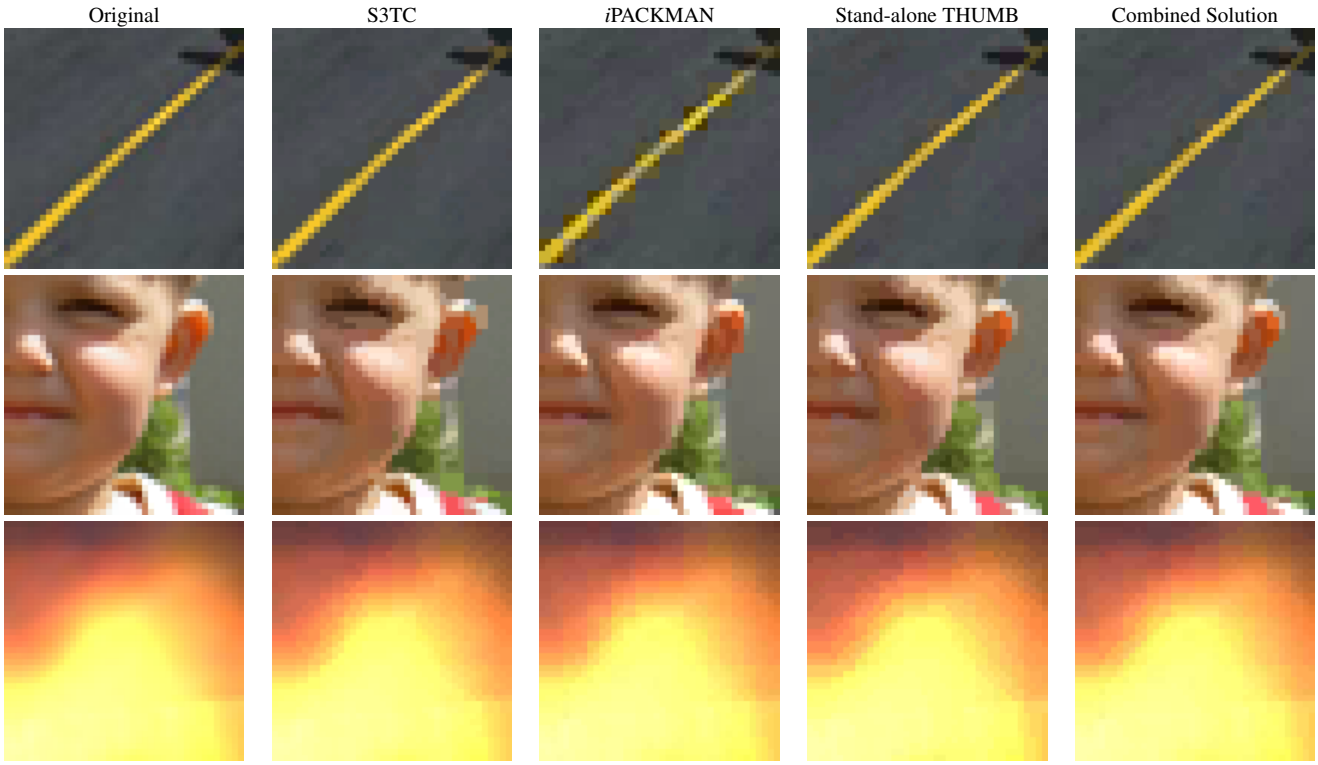


Figure 10: In this figure three examples of typical problem blocks are illustrated. Top: In this example, which shows a road, the main problem of *i*PACKMAN can be seen. Subblocks containing two distinct hues are coded poorly. Middle: Here is an example of the strength of *i*PACKMAN. Small transitions in luminance are coded well as can be seen around the eye and on the cheek. THUMB and S3TC have a more blocky appearance. A large image artifact can also be seen in the ear for S3TC. Since the combined solution inherits the strength of *i*PACKMAN, these blocks are coded well. Bottom: This last example is a cut-out of an explosion. Here S3TC performs the best thanks to its linear interpolation between the base colors. As the blocks contain more than one hue, the result tends to be blocky for *i*PACKMAN. THUMB encodes the image a little bit better even though some edges can be seen.

Lighting Effects for Mobile Games

Jeppe Revall Frisvad* Niels Jørgen Christensen† Peter Falster‡

Informatics and Mathematical Modelling
Technical University of Denmark

Abstract

Adapting games to the miniature screens and limited processing power of mobile phones is quite a challenge. To accommodate these technical limitations, mobile games are often two-dimensional (2D), tile-based, and seen from above. Such games rarely present much creativity with respect to dynamic lighting. Textures are merely plastered onto the tiles. A recent game release has, however, shown that effects such as dynamic light sources can spice up a mobile game of this type quite a bit. In this paper we carry the idea of dynamic lighting effects for tile-based 2D games even further. In particular, we present simple and efficient techniques for shadows and fluctuating fog which can greatly improve the gamer's visual experience.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: lighting effects, mobile games, shadows, fog synthesis, real-time.

1 Introduction

Mobile games rarely feature dynamic lighting effects. An obvious reason is the limited processing power available. Another reason is the different gaming experience that one encounter when playing a game on the small screen of a mobile phone. Since the gaming experience is different, the gameplay designed for a mobile game should be adapted accordingly, see [Nok 2003].

On a small screen the player quickly looses track of game characters and objects. A solution is to make games in a style where everything is seen from above. This ensures the best possible overview. It, therefore, seems reasonable to assume that two-dimensional, tile-based games viewed from above always will be relevant on the miniature screens of mobile phones. For that reason, we propose some simple methods enabling dynamic lighting effects in games of this type.

Quite recently the mobile game Darkest Fear™ from Rovio Mobile was shipped as the first mobile game marketing itself on its ability to incorporate dynamic lighting effects in its gameplay, see [Rov 2005]. And it received an Airgamer Award from Germany's leading mobile game reviewer with the review punch line: "Innovative and exciting!" [Biedermann 2005]. This strongly indicates that lighting effects are worth the effort in this category of games.

*e-mail: jrf@imm.dtu.dk
†e-mail: njc@imm.dtu.dk
‡e-mail: pfa@imm.dtu.dk

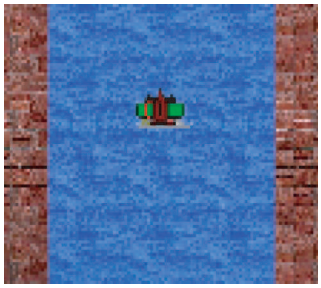


Figure 1: The gamelike scenario we will be working with.

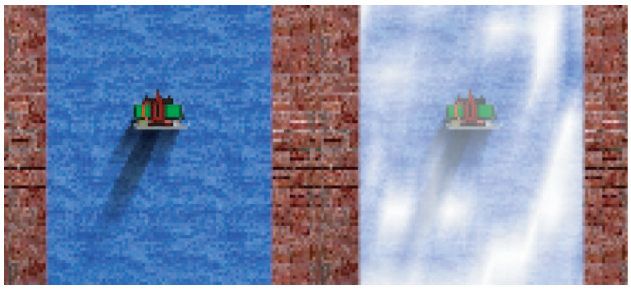


Figure 2: Examples of the lighting effects we will describe efficient methods for in this paper.

Darkest Fear has lighting effects such as movable light sources providing an attenuating illumination of the surroundings. Moreover several light sources are turned on and off dynamically. The shadows of the dynamic characters are merely blobs, which are a part of the sprites[1] associated with a character.

In this paper we suggest a simple technique to do dynamic shadows in tile-based games. Moreover we give a simple technique to create fluctuating fog in such games.

Figure 1 presents the central part of a sample frame from the gamelike scenario we will be working with in this paper. This tile-based demo scenario is compiled for a Windows PC platform, but could as well have been compiled for a mobile phone platform if we had had the necessary development tools at our disposal. The scenery is viewed from above, as we argued is often sensible in mobile games. In Figure 1 no lighting effects have been applied. Compare this example to the images in Figure 2, where the sample frame has been spiced up with the inexpensive lighting effects to be described in the following sections. Hopefully the reader agrees that such effects make the scene more interesting.

The lighting effects we present have, clearly, been done before in games for PCs and consoles. Here the environment is, however, most often three-dimensional which means that the employed shadow and fog algorithms are slightly different from the ones we describe. The types of 2D games which have a third dimension that is always projected on the same plane, give us an option for inex-

[1] A sprite is basically a rectangular pixel map.

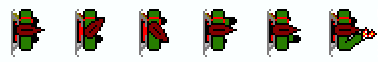


Figure 3: Six sprites for animation of the character in our gamelike environment. The character is a Robin Hood-like person wearing a hat and having a bow over his back pack. He is seen directly from above.

pensive calculation of 3D lighting effects. In the following we will describe how this is done.

2 Lighting Effects

Jim Blinn is one of the pioneers who introduced several lighting effects for three-dimensional environments in the seventies and eighties. Some of his ideas were planar projection shadows, see [Blinn 1988], and rendering of cloud cover, see [Blinn 1982]. Inspired by these ideas, we describe, in the following, how shadows and fog (or cloud cover) effects can be incorporated in two-dimensional games.

The general question to be answered in the following, is how to create a third dimension in an otherwise two-dimensional gaming environment. And, after this information has been provided, how can we construct lighting methods that are not processing intensive. To provide answers, we must take advantage of the two-dimensional nature of the environment.

2.1 Planar Projection Shadows

Consider the sprites for the character of our game, see Figure 3. The sprites have a resolution of 32×32. To find shadows cast by this character, we must provide some height information.

To keep memory costs low, we decided to make two height curves. One along each axis in the character sprites, see Figure 4. Let ℓ denote the number of pixels a height curve covers. In our case $\ell = 32$. A pair of curves could be constructed for each character sprite, but if the difference between the sprites is subtle, there is no need to have more than a single pair of curves for each character.

The curves are positioned above the character and the curve along the axis making the largest angle with the direction towards the light source, is used for calculation of the shadow. Figure 5 illustrates how the shadow outline is found and can be referred to throughout the remainder of this section.

Each light source in the game should have a 3D position:

$$L = (x_L, y_L, h_L)\,,$$

where h_L is the height of the light source above the xy-plane where everything is rendered.

Suppose we let $C = (x_C, y_C)$ denote the position where the center of the sprite will be drawn next. The 2D direction $\boldsymbol{d}_L$ towards a light source is then given as:

$$\boldsymbol{d}_L = (x_{\boldsymbol{d}}, y_{\boldsymbol{d}}) = (x_L, y_L) - C\,.$$

To decide which curve we want to use for the shadow (**a** or **b** in Fig. 4), we can calculate:

$$\cos\phi_1 = \frac{\boldsymbol{d}_L \cdot \boldsymbol{e}_x}{\|\boldsymbol{d}_L\|} = \frac{x_{\boldsymbol{d}}}{\sqrt{x_{\boldsymbol{d}}^2 + y_{\boldsymbol{d}}^2}}\,, \qquad (1)$$

where $\boldsymbol{e}_x = (1, 0)$ is the direction of the x-axis.

When $\phi_1 \in [\frac{\pi}{4}, \frac{3\pi}{4}]$ or $\phi_1 \in [\frac{5\pi}{4}, \frac{7\pi}{4}]$ we want to use curve (**a**) otherwise curve (**b**). This corresponds to saying that when $\cos\phi_1 \in [-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$ we use curve (**a**) otherwise curve (**b**). Note that there

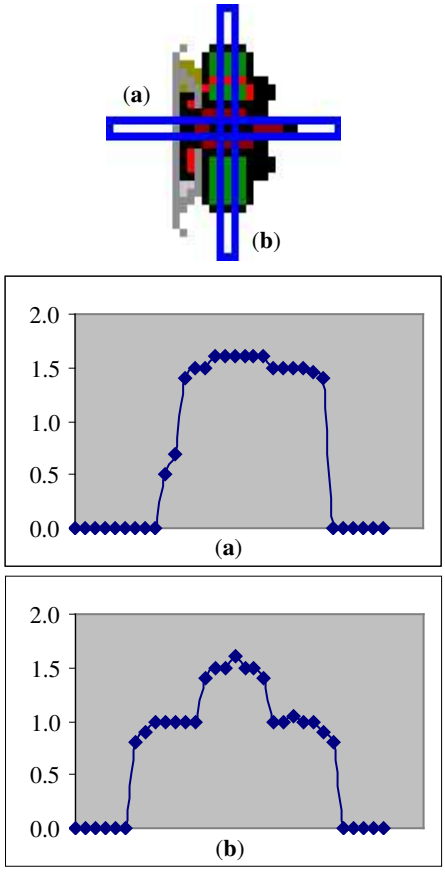


Figure 4: An illustration of the height curves for our character.

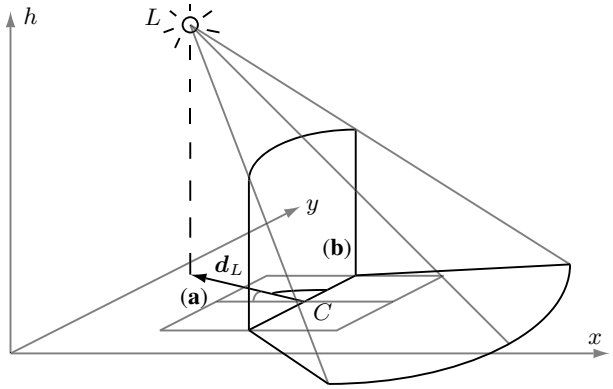


Figure 5: A sketch illustrating how the shadow outline is found. Here curve (**b**) is the chosen curve, since it has the largest angle with the direction towards the light source.

is no need to find the angle ϕ_1 itself as long as the sprites are not being rotated.

It is, however, desirable to rotate the sprites, since then we only need to store them facing a single direction. In our game, an angle $\phi_0 \in [0, 2\pi]$ specifies how much the character sprite has been rotated around its center[2]. This complicates the matter a bit. To find out which curve to use in this case, we calculate

$$\phi = \phi_0 - \text{sign}(y_{\boldsymbol{d}}) \cos^{-1}\left(\frac{x_{\boldsymbol{d}}}{\sqrt{x_{\boldsymbol{d}}^2 + y_{\boldsymbol{d}}^2}}\right), \tag{2}$$

and again, if $\phi \in [\frac{\pi}{4}, \frac{3\pi}{4}]$ or $\phi \in [\frac{5\pi}{4}, \frac{7\pi}{4}]$, we use curve (**a**) otherwise curve (**b**).

Note, in (2), that $\cos\phi_1$ does not supply us with sufficient information to determine ϕ_1. We need to consider the sign of $y_{\boldsymbol{d}}$ to get $\phi_1 \in [0, 2\pi]$. It is also important to realize that ϕ_1 should be subtracted from ϕ_0. This follows since ϕ is the angle between the local x-axis of the sprite and the direction towards the light source. The local x-axis has been rotated to have the angle ϕ_0 with the global x-axis and ϕ_1 is the angle between the global x-axis and the direction towards the light source, hence, $\phi = \phi_0 - \phi_1$.

Depending on whether the character sprites are being rotated in the game or not, either (2) or (1) determines which curve to be used for the projection of the shadow. Each curve should have a direction vector and an origin. The direction of curve (**a**) is $\boldsymbol{d}_{\mathbf{a}} = (w_T/\ell, 0)$ and the direction of curve (**b**) is $\boldsymbol{d}_{\mathbf{b}} = (0, h_T/\ell)$, where w_T and h_T are, respectively, the width and height of a tile in world coordinate units[3]. The needed origin, either $P_{\mathbf{a}}$ or $P_{\mathbf{b}}$, is found according to the current position of the sprite.

Let $\boldsymbol{d}$ and P denote the direction and origin, respectively, of the chosen curve. If the sprite has been rotated, $\boldsymbol{d}$ and P should be rotated accordingly. Each height value on the chosen curve now has a corresponding position:

$$(x_{0,i}, y_{0,i}) = P + i\,\boldsymbol{d} \quad , \quad i = 0, \ldots, \ell - 1\,.$$

Combining the $(x_{0,i}, y_{0,i})$-position with the height value $h_{0,i}$ stored for each position on the curve (see again Fig. 4) gives 3D points $P_{0,i} \in \mathbb{R}^3$, $i = 0, \ldots, \ell - 1$, along the curve. When those have been established, the direction $\boldsymbol{d}_i$ from the light source to P_i is found as

$$\boldsymbol{d}_i = P_{0,i} - L$$

and intersection with the $\boldsymbol{xy}$-plane can be calculated. This is a simple calculation, since we are in possession of the parametric equation of the line with origin at the position of the light source, L, and direction $\boldsymbol{d}_i$ through the point on the curve $P_{0,i}$:

$$(x_i, y_i, h_i) = L + t_i\,\boldsymbol{d}_i\,. \tag{3}$$

When $h_i = 0$, the line intersects the xy-plane. Hence, we quickly discover that

$$0 = h_L + t_i(h_{0,i} - h_L) \quad \Leftrightarrow \quad t_i = \frac{h_L}{h_L - h_{0,i}} \tag{4}$$

finds the value of t_i which, inserted in (3), gives the projection of the ith point on the height curve to the xy-plane in the direction away from the light source. In other words, the points

$$(x_i, y_i) = (x_L, y_L) + \frac{h_L}{h_L - h_{0,i}}(x_{0,i} - x_L, y_{0,i} - y_L) \tag{5}$$

and $(x_{0,i}, y_{0,i})$ with $i = 0, \ldots, \ell - 1$, defines the outline of the shadow in the xy-plane (see again Fig. 5).

[2]In fact we use integers and degrees (from 0° to 359°) for the angles not radians, but that is not essential for the description of the concept.

[3]It is sensible and, of course, convenient to let $w_T = h_T = 1$.

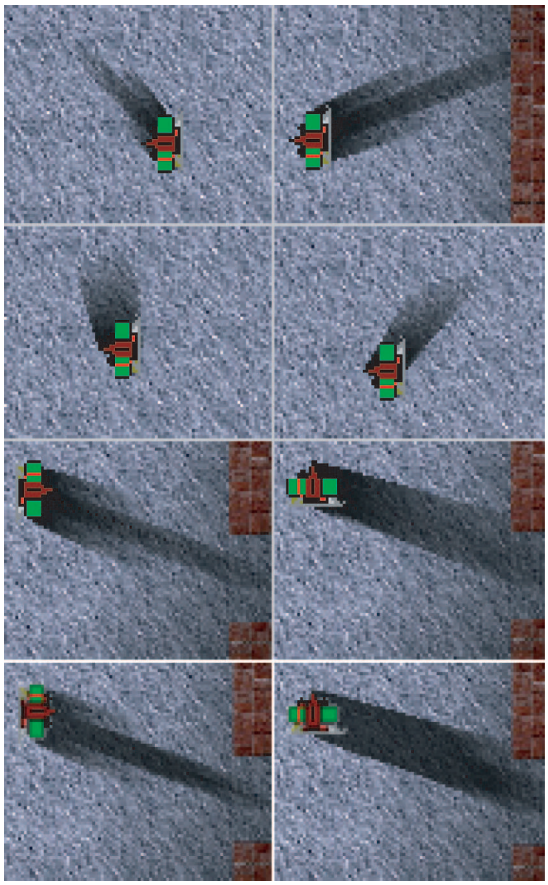


Figure 6: Shadows resulting when the described technique is employed in our case study. In the first three rows the shadow has been rendered with alpha values interpolated over a single triangle strip. For comparison the last row has been rendered using two triangle strips: One representing the umbra region and one representing the penumbra region. The first with constant alpha value, the second with interpolated alpha values. While the difference is subtle, the advantage of the second approach is that we can control the size of the penumbra region.

Having the shadow outline, the shadow can be visualized in many different ways. With a slight abuse of notation let $P_{0,i} = (x_{0,i}, y_{0,i})$, and let $P_i = (x_i, y_i)$. We simply draw the shadow as a triangle strip using the points $P_{0,i}, P_i, P_{0,i+1}, P_{i+1}, \ldots, P_{0,\ell-1}, P_{\ell-1}$. To give a touch of softness to the shadows, we make the points P_i quite transparent ($\alpha = 0.1$) while the points $P_{0,i}$ are left opaque ($\alpha = 1$). Standard linear interpolation of the alpha values between the vertices of each triangle (Gouraud shading, see [Gouraud 1971]) is performed[4] when the triangle strip is rendered. Examples from our case study of the resulting shadows are given in Figure 6.

Linear interpolation of the alpha values is physically incorrect, but seems visually more pleasing than a hard shadow. If distinguishable umbra and penumbra regions are desired, it is necessary to render the shadow as two triangle strips. First we project the height curve to a plane which is moved a constant c closer to the light source. The constant determines the size of the penumbra regions and the shadow outline found at the plane moved closer to the light source will describe the umbra region (the idea to move

[4]In OpenGL and OpenGL ES, Gouraud shading happens automatically when the smooth shading model has been chosen.

the intersection plane closer to the light source was proposed by Gootch et al. [1999] for rendering of planar soft shadows in 3D environments). The projected points $P_{u,i} = (x_{u,i}, y_{u,i})$ are found using (3) with $t_{u,i} = t_i - c$ when $t_i - c \geq 1$ otherwise $t_{u,i} = 1$. Now the points $P_{0,i}, P_{u,i}, \ldots, P_{0,\ell-1}, P_{u,\ell-1}$ can be used to draw the triangle strip for the umbra region with a constant alpha value, eg. $\alpha = 0.75$.

The second strip is drawn using the points $P_{u,i}, P_i, \ldots, P_{u,\ell-1}, P_{\ell-1}$, where the points P_i are the ones found in (5). The points $P_{u,i}$ should be rendered with the same alpha value as before, but the points P_i should be rendered with $\alpha = 0$. A few examples of the shadows resulting from this two strip method are given in Figure 6 for comparison with the method using one strip.

2.2 Fluctuating Fog

The second lighting effect that we would like to describe a simple rendering technique for, is fog and similar semi-transparent phenomena. One way to do fog is simply to have a texture with varying shades and transparencies (alpha values). The texture is then spread over the tiles that should be foggy. This is, however, expensive with respect to memory storage and the result is a static fog which does not give much feeling of realism.

Instead we suggest that a height field composed of a 16×16, 10×10, or an even smaller grid of height values should be sufficient to generate an interesting fluctuating fog.

Suppose the fog is to be spread across a rectangular area in the game specified by three 2D points: Q_0, Q_{top}, and Q_{right}. Let ℓ_x and ℓ_y denote the resolution of the height field grid. Then the vectors used to step through the height field are

$$\boldsymbol{v}_x = \frac{Q_{\text{right}} - Q_0}{\ell_x} \quad \text{and} \quad \boldsymbol{v}_y = \frac{Q_{\text{top}} - Q_0}{\ell_y} .$$

The 2D position of each vertex in the height field is then found as

$$(x_{ij}, y_{ij}) = Q_0 + i\,\boldsymbol{v}_y + j\,\boldsymbol{v}_x ,$$

where $i = 0, \ldots, \ell_y - 1$ and $j = 0 \ldots, \ell_x - 1$. This is combined with the values h_{ij} available in the height field to obtain the 3D positions $Q_{ij} = (x_{ij}, y_{ij}, h_{ij})$.

To have our fog change appearance depending on the viewpoint of the player (that is, to create fluctuations), we need normal information as well as positions. For each vertex in the height field we can calculate two basis vectors $\boldsymbol{b}_{1,ij}$ and $\boldsymbol{b}_{2,ij}$ using one neighboring point in the x direction and one neighboring point in the y direction. Normalized cross products of the two basis vectors at each vertex then provides the normals:

$$\boldsymbol{n}_{ij} = \frac{\boldsymbol{b}_{1,ij} \times \boldsymbol{b}_{2,ij}}{\|\boldsymbol{b}_{1,ij} \times \boldsymbol{b}_{2,ij}\|} .$$

Depending on the processing power available versus the memory available, normals or positions could be stored in memory or recalculated as one thinks fit. If the fog is supposed to move around in the game, the positions must, of course, be recalculated, but the normals could still be kept static. Normals should be recalculated if the height values change.

To render the fog, the position of the eye V must be available. In our game, the eye is positioned directly above the controlled character which is always centered. Centering the character controlled by the player is one of the good advises in [Nok 2003]. This gives the player a better chance to follow the game on a small screen.

The angle θ_{ij} between the direction towards the viewpoint, V, and the normal, $\boldsymbol{n}_{ij}$, at each vertex is now used to attenuate the light transmitted directly through the fog. The formula calculating attenuation of directly transmitted light is well known, see eg. [Chandrasekhar 1960]. Assuming that the fog medium has a constant extinction coefficient σ_t throughout, and that the fog goes all the way to the xy-plane, the attenuation computation is given as

$$\alpha_{ij} = e^{-\tau_{ij}/|\cos\theta_{ij}|} , \tag{6}$$

where $\tau_{ij} = \sigma_t h_{ij}$ is the optical depth of the fog below the vertex. We do not have to worry too much about the exact meaning of the extinction coefficient in this context. It suffices to think of σ_t merely as a scale: When it increases the fog becomes more dense. To calculate $\cos\theta_{ij}$ we have

$$\cos\theta_{ij} = \frac{V - Q_{ij}}{\|V - Q_{ij}\|} \cdot \boldsymbol{n}_{ij} . \tag{7}$$

Left to find is a shade for the fog at each vertex. In our opinion the height values scaled such that they live in $[0, 1]$ gives acceptable grey shades for a fog in dark surroundings.

The fog is now ready to be rendered on top of the tiles it was spread across. To do the blending of the fog with the scenery below it, we call the grey shade, found for each fragment as an interpolation of the scaled height values, the *source* and denote it L_{src}. The *destinations* are the existing colors L_{dst} of the fragments. The blending should be done such that

$$L_{\text{blend}} = L_{\text{src}} + \alpha_{\text{src}} L_{\text{dst}} .$$

Some resulting images from our case study are presented in Figure 7. If we want a white fog, corresponding to a cloud cover lit from above by the sun, we do not use grey shades, but set all color values (except the alpha value, of course) to 1 and use the following function for blending:

$$L_{\text{blend}} = (1 - \alpha_{\text{src}}) L_{\text{src}} + \alpha_{\text{src}} L_{\text{dst}} .$$

An example of the result is shown in Figure 2.

3 Dealing with Mobile Phone Limitations

Battery life is a main concern on all mobile devices. Floating point processors consume more power than integer processors, therefore most mobile devices (even high-end devices) have no floating point processor [Astle and Durnil 2004, Chap. 6]. Instead floating point calculations are simulated in software and that is too slow for game programming. How can we implement the lighting effects described in the previous section with no floating point operations? The answer is to employ fixed point arithmetics. Michael Street [2004] gives an excellent primer to fixed point arithmetics describing efficient implementations of all the basic math operations including square roots, vector normalization, and trigonometric functions (code is included).

The trigonometric functions are not really used in the calculations for our lighting effects, since cosine of an angle is found using the dot product between two vectors as shown in (1) and (7). If a game uses rotation of sprites as we do in our demo, it is necessary to evaluate (2) where an arcus cosine ($\cos^{-1}$) is employed. Moreover evaluation of the exponential function (e) is needed for calculation of fog transparency in (6). We recommend that look-up tables are used for fixed point evaluation of these two functions.

While it is straight forward to construct a look-up table for arcus cosine, since $\cos^{-1}(x)$ only takes values $x \in [-1, 1]$ as argument, it is not immediately obvious how to construct a look-up table for the exponential function. In our implementation we chose a look-up table with twenty entries for evaluation of e^{-x} when $x \in [0, 1)$,

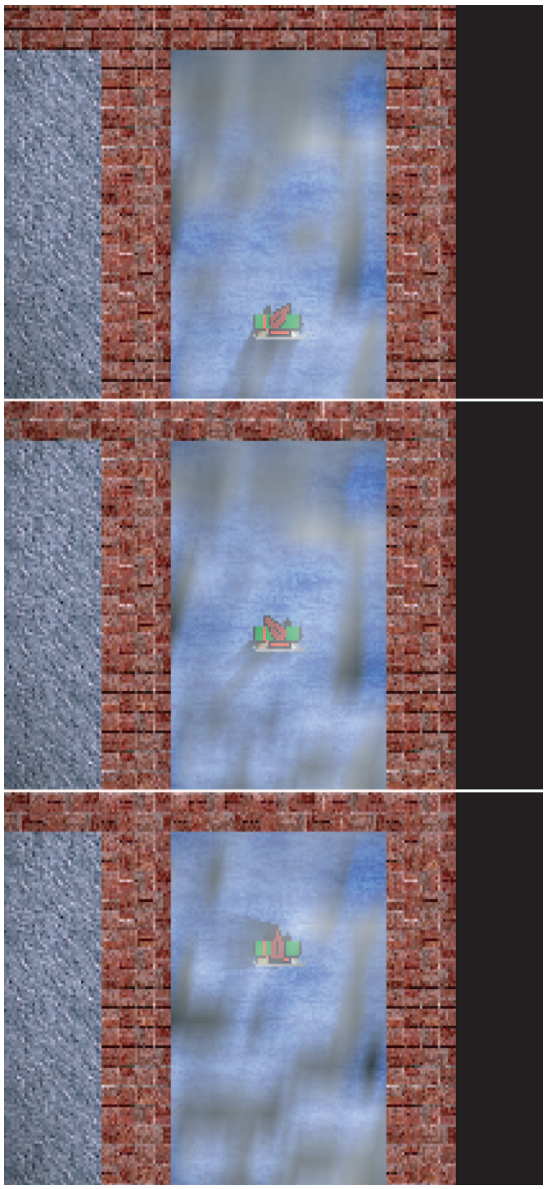


Figure 7: A few screen shots to illustrate that the fog changes appearance as the eye point moves around above it. The eye point is always placed directly above the character.

we have ten entries for $x \in [1, 2)$, and five entries for $x \in [2, 3)$. If $x \in [3, 10)$, the function is approximated by a straight line:

$$f(x) = \frac{e^{-3}}{7}(10 - x)\,.$$

Finally, when $x \in [10, +\infty)$, the return value is set to zero.

To make sure that our demo can potentially run on a mobile device, we have made an implementation using the PowerVR MBX OpenGL ES 1.1 SDK for desktop PCs[5]. PowerVR even supply a look-up table for fixed point evaluation of arcus cosine as a part of their OpenGL ES Tools library.

Both the shadow and fog described in the previous section can and should be drawn using triangle strips. This is efficient and suitable for both OpenGL ES and the J2ME 3D graphics API.

[5]The SDK is available at http://www.pvrdev.com/Pub/MBX/OGLES/.

	no shadow	**one strip**	**two strips**
w/o. fog	50.5	47.8	47.5
w. fog	30.1	29.0	28.9

Table 1: Frame rates (frames/sec) measured on an old laptop with or without fog and with no shadow, shadow drawn using a single triangle strip, or shadow drawn using two triangle strips. Please note the differences between the frame rates rather than the frame rates themselves.

4 Results and Discussion

While there, as indicated in the previous section, are many implementation constraints when games are written for mobile phones, see also [Coulton et al. 2005], processing power is rarely an issue. The reason is that smaller screen size means "fewer pixels to push with each buffer flip" [Nok 2003]. Nokia also argues that the 104 MHz speed of the average mobile phone should be sufficient to support real-time rendered 3D graphics to a limited extent. This makes us confident that at least the shadows we have described are very well suited for mobile games. The fog may be slightly over the top. On the other hand, we only need to recalculate the terms in the part of the fog which is visible.

To give a feeling of the performance hit entailed by the described methods, we have simulated our case study on a 400 MHz Pentium3 laptop. The simulations were run in a 250×250 resolution and with a fog grid of 16×16 vertices. What is important is not the frame rates themselves, but rather the difference between them. The program could run much faster in a lower screen resolution and with textures of a lower resolution, but then it would be difficult to tell the difference in performance between eg. shadow and no shadow since frame rates become quite unstable when they are high. The frame rates from our experiments on the laptop are given in Table 1.

The performance hit of the fog is seemingly a little high. The calculations described in the previous section are, however, not the expensive part of the fog rendering. They reduce the frame rate only by a frame or two. The expensive part is the Gouraud interpolation of colors and alpha values across the triangles that are drawn. Hence the larger the number of pixels which are covered by fog, the larger the performance hit. The white fog shown in Figure 2 is, in fact, less expensive to render than the fog with grey shades, since in the second case, the color values have to be interpolated as well as the alpha values. With the emergence of mobile GPUs (see some of the possibilities they entail in [Macedonia 2004]) which all have a hardware implementation of the Gouraud shading, the fog synthesis we describe should become very feasible for mobile games.

Considering the images in Figure 6 more closely, we will discover that the described shadows have some limitations. While the shadow outline is projected correctly onto the xy-plane, the method does not account for eg. the gap between the legs of the character. The gap is not captured by the shadow outline we can construct using a single height curve. To render such details, additional height curves (below the existing ones) would have to be incorporated.

Another problem is that the shadows are *always* projected onto the xy-plane. Suppose there is a wall next to the character, then the shadow will end up on top of the wall. This is definitely not desirable. A simple way to work around the problem is to draw the walls (and rooms on the opposite side of the wall) after the shadow has been rendered. This trick does not really fix the problem, the shadow may still appear on the opposite site of a closet or some other object close to the character. But then, on the other hand, the shadow caused by the object close to the character would usually cast a shadow itself on top of the shadow from the character. Hence, the problem is not fatal.

Throughout the main part of this paper we have referred to tile-

based games viewed from above as the type of games where the described methods can be employed. There is, however, nothing to prevent us from using the same methods in games which are not tile-based or games which are not viewed directly from above. As long as the objects are all rendered as sprites moving on the same plane, the described shadow and fog techniques are applicable. It so happens that tile-based games usually belong to the category of games for which our techniques are useful. Therefore we describe our methods as rendering methods suitable for tile-based games.

5 Conclusion

An efficient method for rendering of projection shadows in tile-based 2D games has been presented in this paper. Shadows are found through construction of two height curves above each sprite describing a dynamic object or character in a game. The calculations needed for our shadows are sufficiently simple to allow for implementation on mobile phones. The shadows are not exactly physically correct, but they are a great improvement as compared to no shadows or simple blobs beneath the characters.

Additionally we describe a method for creation of fluctuating fog. The fog is described by a height field placed above a plane and fluctuates as the eye point moves around in the scene. The changing transparency of the fog is calculated according to physical considerations in the theory of radiative transfer. The shade of the fog is, however, simplified to limit the performance hit of the fog rendering. The OpenGL ES software simulation of Gouraud interpolation across the triangles visualizing the fog has shown to be the limiting factor with respect to processing power. With the emergence of mobile GPUs, this problem will disappear. In our opinion, the fog resulting from the rendering method we describe, is quite convincing. In particular we find that the fluctuations are an important part of the conviction. If the fog is rendered as a static semi-transparent layer, the gamer will hardly get any feeling of visual realism.

In light of recent development in and attention to the mobile gaming market[6], we believe that the time is right for incorporation of more lighting effects in mobile games. Hopefully this paper will help the gaming companies getting started on projection shadows and fluctuating fog.

Acknowledgement

Thanks to Rasmus Revall Frisvad for letting us use the character sprites which were created approximately ten years ago for a garage game project.

References

ASTLE, D., AND DURNIL, D. 2004. *OpenGL® ES Game Development*. Thomson Course Technology PTR, Boston, MA.

BIEDERMANN, S., 2005. Darkest fear: Bringen sie licht ins dunkel! Airgamer, July. http://www.airgamer.de/.

BLINN, J. F. 1982. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics (SIGGRAPH '82 Proceedings) 16*, 3 (July), 21–29.

BLINN, J. F. 1988. Me and my (fake) shadow. *IEEE Computer Graphics and Applications 8*, 1 (January), 82–86.

[6]The September 2005 issue of Game Developer Magazine was devoted entirely to the subject of mobile gaming.

CHANDRASEKHAR, S. 1960. *Radiative Transfer*. Dover Publications, Inc., New York. Unabridged and slightly revised edition of the work first published in 1950.

COULTON, P., RASHID, O., EDWARDS, R., AND THOMPSON, R. 2005. Creating entertainment applications for cellular phones. *ACM Computers in Entertainment 3*, 3 (July).

GOOTCH, B., SLOAN, P.-P. J., GOOTCH, A., SHIRLEY, P., AND RIESENFELD, R. 1999. Interactive technical illustration. In *Proc. of the 1999 Symposium on Interactive 3D Graphics*, 31–38.

GOURAUD, H. 1971. Computer display of curved surfaces. Tech. Rep. UTEC-CSc-71-113, Department of Computer Science, University of Utah, June. Also in *IEEE Transactions on Computers*, vol. C-20, pp. 623–629, June 1971.

MACEDONIA, M. 2004. Small is beautiful. *Computer 37*, 12, 122–129.

NOKIA CORPORATION. 2003. *Designing Single-Player Mobile Games*, September. Version 1.01.

ROVIO MOBILE. 2005. *Darkest Fear: Fact Sheet*. http://www.rovio.com/.

STREET, M. 2004. A fixed point math primer. In Dave Astle and Dave Durnil: *OpenGL® ES Game Development*. Thomson Course Technology PTR, Boston, MA, ch. 4, 67–88.

Tangible User Interface for Chemistry Education: Visualization, Portability, and Database

Joakim Almgren [1], Richard Carlsson [1], Henrik Erkkonen [1], Jonas Fredriksson [1], Sanne Møller [1], Henrik Rydgård [1], Mattias Österberg [1], *Kristina Bötschi [2], **Benedikt Voegtli [3], Morten Fjeld [4]
TableTopInteraction Lab (http://t2i.se/) at IDC, CSE, Chalmers University of Technology, Sweden
*Department of Psychology, University of Zurich, Switzerland, ** HyperWerk, FHBB, Switzerland

Abstract

Augmented Chemistry (AC) is an application that utilizes a tangible user interface (TUI) for organic chemistry education. Based on the outcome of an extensive evaluation, we are in the process of extending the AC system[5]. Firstly, for enhanced interaction, keyboard-free system configuration, and internal/ external database (DB) access, a graphic user interface (GUI) has been incorporated into the TUI. Three-dimensional (3D) rendering has also been improved using shadows and related effects, thereby enhancing depth perception. Secondly, AC has been ported to different operating systems and is now compatible with Linux-, Windows-, and Mac OS X based platforms. This enables the use of a wider range of hardware: USB and Firewire (IEEE1394) cameras are now supported. Finally, system capacity to import and visualize molecules from an extensive XML-based DB has been realized. This gives users the ability to download and interact with any molecule up to a certain complexity.

Keywords

Augmented Reality, Tangible User Interface, Education, Organic Chemistry, Octet Rule, GUI, TUI

CCS

Input devices; Three-dimensional displays; Display algorithms; Viewing algorithms; Virtual device interfaces; Curve, surface, solid, and object representations; Color, shading, shadowing, and texture; Virtual reality

1. Introduction

Augmented Chemistry (AC) was developed at HyperWerk FHBB[6] [Voegtli, 2002; Fjeld and Voegtli, 2002]. It was extended with additional functionality and evaluated in a joint project which involved ETH[7], HyperWerk FHBB, and the aprentas[8] school of chemistry [Fjeld et al., 2004]. As part of the project, an empirical evaluation of AC was conducted by Bötschi [Bötschi, 2005], who studied how AC compares to the traditional ball-and-stick method of learning organic chemistry. Subjective preferences of the two alternative systems was one of the many variables measured (Fig. 1). On the basis of the first version and Bötschi's findings, we have further extended the AC system addressing three objectives:

- improved visualization
- extended portability
- ability to import from public chemistry DBs

All three objectives called for further implementation in the AC system. Aiming for improved comfort of use, ease of use, and ease of learning the system [Bötschi, 2005] (Fig. 1), we established a set of goals and informal requirements. Following these, visualization has been significantly improved utilizing shadows and luster to facilitate user comprehension of complex 3D models and information. (See the included video documentation[5]). Secondly, to increase the potential software and hardware compatibility, portability was extended. The new import functionality from public chemistry DBs offers users an expanded library of molecules for viewing and manipulation. Finally, for enhanced interaction, keyboard-free system configuration, and internal/external DB access, a GUI was incorporated into the TUI.

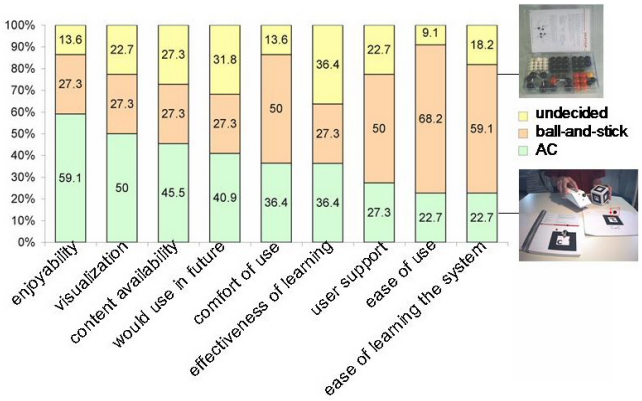


Figure 1: Test subjects' mean preferences for AC (bottom), ball-and-stick (middle), or undecided (top). AC ranked higher in enjoyability, visualization, content availability, future use, and effectiveness of learning. AC ranked lower in comfort of use, user support, ease of use, and ease of learning the system [Bötschi, 2005].

2. Tangible user interface (TUI)

AC uses a TUI enabling its user to compose and directly interact with 3D molecular models. The system was designed to assist in teaching abstract organic chemistry concepts such as molecular forms, the octet rule, and bonding. Following the conventional implementation of the AR Toolkit [Kato et al., 2000], physical tools carry one or more fiducial markers, connecting each tool to an animated 3D model so that both the tool and the model can be seen in a composite image.

[1] {vilgon, cri, erkkonen, jonafred, traco, tronic, osterber} @dtek.chalmers.se
[2] kristina@gehring.biz
[3] b.voegtli@hyperwerk.ch
[4] morten@fjeld.ch
[5] Video: http://www.t2i.se/pub/video/AC_SIGRAD2005.avi
[6] HyperWerk FHBB: http://www.hyperwerk.ch/
[7] Eidgenössische Technische Hochschule Zürich: http://www.ethz.ch/
[8] aprentas: http://www.aprentas.com/

The system consists of the booklet, the gripper[9], the cube, the platform, a camera, and software. The booklet contains one element per page, each with its name and relevant information (Fig. 2). By using the gripper, users can pick up elements from the booklet and add them to the molecule in construction on the platform (Figs. 3, 4).

Figure 2: The booklet, the gripper, the cube, and the function cards (clockwise from top left).

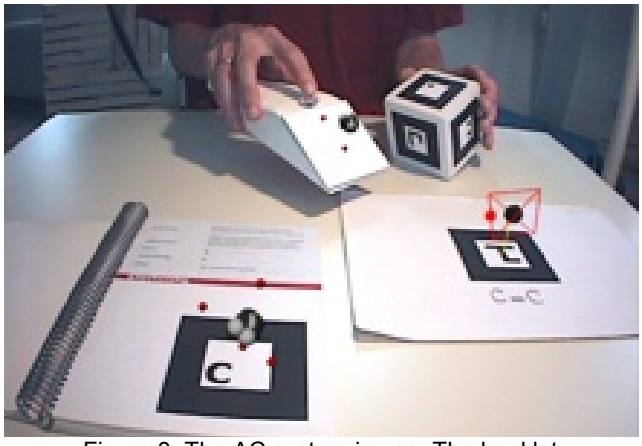


Figure 3: The AC system in use: The booklet, the gripper, the cube and the platform (left to right).

By rotating the cube, the user rotates the molecule, which determines how and where the new element shall bond. The function cards represent specialized functions which are activated when drawn onto the platform. These include the browser, the tag-toggle, the cleaner, the benzene-template, and the dipole [Fjeld and Voegtli, 2002; Fjeld et al., 2004]. While optical tracking systems enabling smaller markers and function cards do exist and would free up tabletop real estate[10], their source is not yet open. At the same time, we did not consider utilizing more cards for system configuration and internal/external DB access because of limited hardware real estate. Consequently, the limited physical space triggered the idea of integrating a GUI into the TUI. The idea of combining TUI with GUI has been explored in other projects [Ishii et al., 2002].

[9] The gripper first consisted of a wireless mouse rebuilt into a cage with a marker. Now, the marker is simply fastened to a conventional wireless mouse.

[10] ARTag, http://www.cv.iit.nrc.ca/research/ar/artag/, is not available under a open/free license.

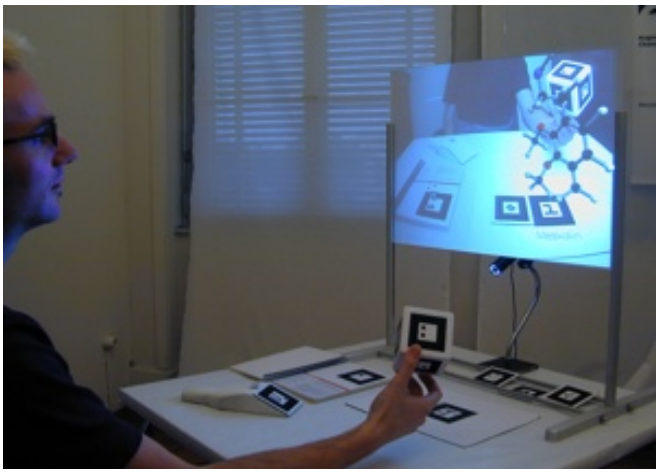


Figure 4: The AC system running on a standard workstation with a frame grabbing camera. USB and FireWire cameras are equally supported.

3. GUI and TUI, dual mode, 3D rendering

Here, we present realizations where particular attention is paid to integration of a GUI into the existing TUI. Firstly, we will explain the benefits of a GUI to support system configuration and, internal/external DB access. Secondly, we will show how dual mode presentation of the learning content was realized. Finally, we will present the 3D display and rendering issues and then the consequential improvements.

3.1. GUI and TUI: design issues

User studies showed that a problem in using the AC system – as compared to a traditional ball-and-stick method – was that controlling system settings often obstructed the learning process [Bötschi, 2005]. Many configuration settings are more suited to a GUI such as molecule size, element labeling, and system parameters. Such settings were initially mapped onto keyboard function keys. However, many users had difficulty using function keys while keeping complex chemical models in mind. The importation of molecules from external DBs also made it necessary to visualize large DB lists of molecules. So, to simplify interaction with system configuration settings and the new DB functionality, a purely mouse-controlled GUI was integrated into the primary TUI. Consequently, the GUI allows for a much more user-friendly system. Plus, keyboard-free operation allows for more efficient use of tabletop real estate.

In order to assure portability, the GUI was realized using OpenGL. To avoid a one-to-one reclaim of screen real estate, we designed a GUI with a permanent button in the corner that activates a pop-up menu with each alternative activating a graphic overlay dialogue box (Figs. 5-10).

Since OpenGL was already being utilized in AC for video image display and molecule rendering, it was the only application programming interface (API) feasible for GUI drawing. OpenGL has an adequate capability for high-quality GUI drawing because of its fast hardware acceleration. It is significantly faster than X or Win32 GDI when running on 3D-capable hardware. Alpha-blended texture mapping makes it possible to draw not only well designed 3D objects, but also smooth 2D windows and buttons.

And a normal painting program is all that is required to customize the appearance in detail.

Alpha-blending also made it possible to easily implement the GUI's unique fading feature. When users click on the AC background video image, the entire GUI fades, allowing for a better view (Fig. 8). Then, when clicking anywhere on the translucent GUI, it regains full visibility. This effect would have been practically impossible with most traditional windowing APIs.

Since there was no suitable, light-weight GUI toolkit using OpenGL available under the GPL license, we implemented a toolkit designed in-house. We foresee to base future GUI development on the current achievements (Figs. 5-9).

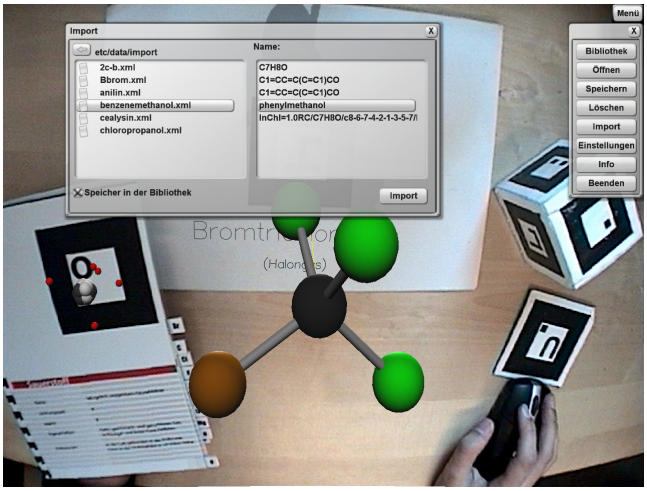


Figure 5: The GUI's import function from an external DB.

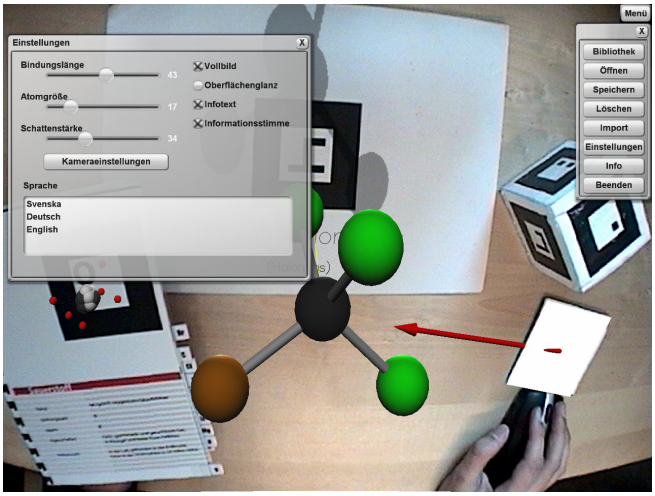


Figure 6: The GUI's configuration menu.

The GUI's graphical overlay is utilized for the import function (Fig. 5), system configuration (Fig. 6), and as a browser for the internal molecule DB (Fig. 7). All the molecules loaded into the system, whether predefined or imported from external DBs, are indexed by their chemical name (Fig. 5). This allows users to quickly browse through the molecules and select one by clicking its name. The browser then displays the selected molecule as a simplified 3D representation (Fig. 7).

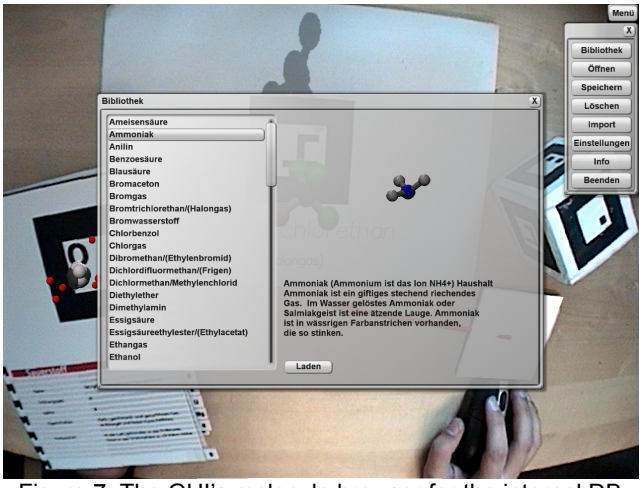


Figure 7: The GUI's molecule browser for the internal DB.

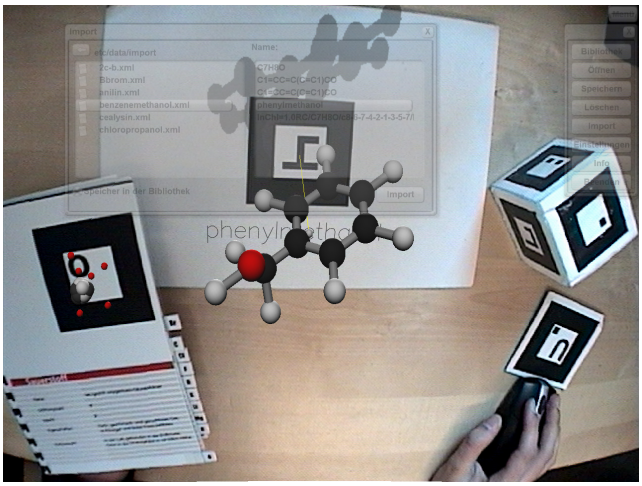


Figure 8: The focus shift from the GUI to the TUI.

3.2. Dual mode: textual and aural information

Evaluations demonstrated that there was a need to improve the presentation of the educational texts. Students have different learning styles, some preferring aural methods and some visual [Bötschi, 2005]. The first versions of AC did not take this into account. In these versions, when a predefined molecule was loaded or a constructed molecule was recognized, audio information was output. One problem with this was the inability to rewind or replay the audio information. To enhance user control of the information flow we are developing a dual mode system which would include a graphical overlay with the information in textual form. This display of educational text has been partly realized (Fig. 9). The realization of a user option between textual display, audio output, and both is foreseen.

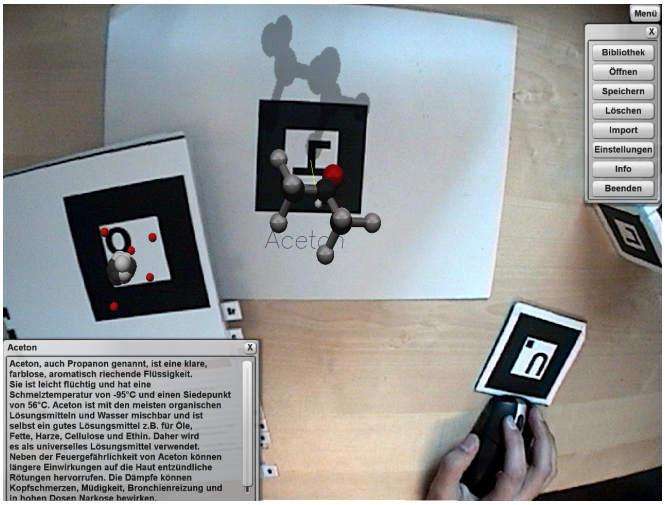


Figure 9: Educational text is displayed.

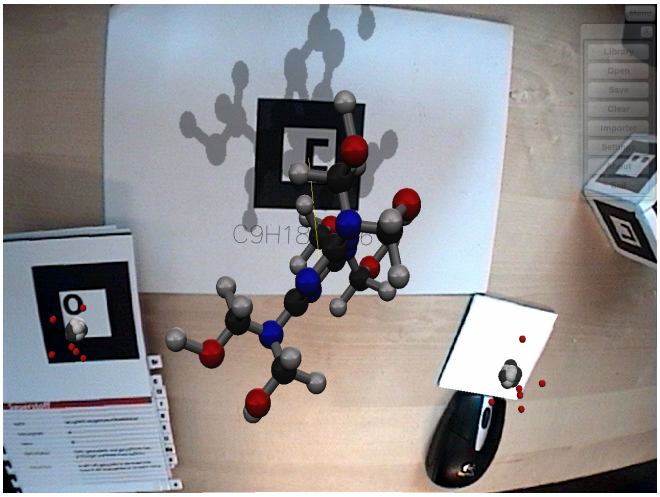


Figure 10: Molecule shadow and luster rendering; the use of shadows may potentially be expanded for projecting information on multiple planes.

3.3 Improved 3D visualization and rendering

Molecule visualization is being iteratively improved and evaluated. The user study [Bötschi, 2005] showed that manipulating the structure of the molecules using the TUI was relatively difficult. Small mistakes, e.g. misplacement and accidental removal of atoms, disrupted the learning process. To increase the ease of learning and operating the system, we believe that the interface needs to feel more natural and more like the conventional ball-and-stick method, but with the benefits of a TUI. To enhance the appearance of the computer-generated molecules, shadow rendering was added to the graphics engine. The projection of shadows and information on different planes (Fig. 10), lends a sense of order and structure to the complex information. Shadows play an important role in visualizing 3D models by improving the user's depth perception [Shon and McMains, 2004]. This allows users to manipulate the molecules with greater precision, enhancing the TUI experience. However, when viewing more complex molecules, the shadows have little or no added value [van Liere, 2005]. Addressing this issue, the system allows users to change the shadow darkness and turn it on or off easily.

4. Portability

The AC system's portability has been extended, so it is now compatible with a wider range of software and hardware platforms, including laptops. Furthermore, new options and an easy-to-install CD-ROM enable user-friendly installation. Newly enabled for multilingual configuration, the AC system is now accessible to a potentially larger user community.

4.1. Operating systems and cameras

Originally, AC was developed for Linux and only supported video cameras connected through a frame grabber. We have recently ported it for use on Windows and Mac OS X and switched to a newer version of ARToolKit. This allows for operation with a much wider range of hardware, including standard, low-cost USB and Firewire (IEEE1394) cameras. The increased compatibility maximizes the number of potential users and simplifies the use of AC.

4.2 Multilingual configuration

The AC system's first version presented language information exclusively in German. In order to make it accessible to more users, we have prepared it for translation into multiple languages. This required that molecule names, structure information, and educational text/audio be stored in the internal DB, enabling display in multiple languages. So far, the GUI has been translated into English and Swedish. Meanwhile, most of the educational information remains to be translated from German.

5. Ability to import from an external molecule DB

By enabling users to construct and examine organic molecules in 3D, a greater understanding of organic chemistry is expected. However, interviews with subject-matter experts revealed that there was a need to not only visualize and interact with user-defined molecules, but also with predefined molecules.

5.1. Advantages of an external DB

A new system feature is the ability to visualize any pre-defined, existing molecule[11]. While manual assembly of a molecule, atom by atom, is an effective way of learning structural details, it can be wearisome for larger molecules. That is to say, the pedagogical benefits of the TUI may be lost when user attention is drawn more to interaction than learning. We wanted to avoid such undesirable effects and create a more versatile TUI, so access to an external DB of predefined molecules was viewed as advantageous. While the first version of the AC system used its own proprietary format for storing molecule definitions and information, we have newly enabled the system for standard molecule file formats.

However, the import of molecules into the AC system from an external DB is limited to simpler molecules that conform to the octet rule and have a tetrahedron-based structure. In order to construct and visualize more complex organic and inorganic molecules, the current static composition model in AC would have to be replaced by a dynamic model based on molecular mechanics.

[11] There are still limitations in molecule complexity. For example, only one benzene ring can be represented in a molecule.

5.2. Database format

A variety of molecule DBs are available, including Beilstein[12], ISIS[13], and PubChem[14], all of which are used by professionals for many purposes. These DBs differ in many ways, but most importantly in file format. After extensive research into molecule file formats and interviewing subject-matter experts, we chose an XML based format because it is a standard that is easily convertible to non-standard formats. PubChem is just one XML enabled DB containing a vast amount of molecule information, created by the National Center for Biotechnology Information[14]. And by having access to PubChem, users can easily choose to import a molecule of interest, visualize it in 3D, and interact with it through AC.

5.3. Conversion from external to internal data structure

While most standard data formats like PubChem include predefined, spatial offset coordinates for the positioning of the atoms in a molecular model, the AC internal data structure does not. Therefore, AC does not consider molecular structure separately from the positioning of single atoms. Rather, the placement of atoms in a molecule is determined by their composition following a tetrahedral pattern [Fjeld and Voegtli, 2002].

When importing a PubChem molecule into AC, the data is converted into the internal data structure. Currently, the AC system can only process molecular model data that complies with its internal data structure. For example, only molecules with up to one benzene ring can be imported.

6. Discussion and outlook

Based on the AC system's first version and the outcome of its comparative evaluation [Bötschi, 2005], we have implemented a new set of functions and features into the system, as described in this paper. First, we have integrated a GUI into the TUI and improved 3D visualization and rendering. Then, we have extended portability to Windows and Mac OS X, enabling the use of different camera types. Finally, we have made AC compatible with an external XML database. Currently, we are in the process of translating information into languages other than German, supporting the dual mode of aural and textual learning content, and capitalizing on multiple-plane shadow rendering.

To evaluate our work we conducted a small, qualitative user study in which six secondary school students tested the system before and after our implementations. They constructed a set of molecules and then gave their subjective opinions about the system's ease of use, ease of learning the system, and if they would be likely to use it in the future when learning organic chemistry. Most of their opinions of the earlier version coincided with those found in the major user study [Bötschi, 2005]. Their opinions of the later version indicated that we had successfully improved both the system's ease of use and ease of learning the system. Their general opinion was that the additional functionality and added features had increased the probability of their using a similar system in an actual learning situation. We foresee a more extensive usability evaluation of the GUI/TUI integration and the GUI functionality in the future.

A possible improvement for AC would be the implementation of an alternative viewing mode for externally created molecular models from PubChem that do not comply with the internal data structure. Structural comparisons as internally carried out when building new models would not be used in this alternative viewing mode. Hence, a conversion of the PubChem data to the AC internal data structure could be bypassed. We will consider this in the future.

In a related project, researchers at The Scripps Research Institute have added markers to passive, ready-made ball-and-stick structures [Gillet, 2005]. Like in Fjeld et al. [Fjeld et al., 2004], they have visualized electrostatic fields and a local field vector. Going beyond the results of Gillet et al., we foresee the exploration of intelligent, physical balls and sticks. The imagined use would require spatial tracking of individual balls and sticks which would communicate the constructed molecule composition, position, and rotation. Hence, a one-to-one 3D virtual augmentation should become possible, enriching the physical ball-and-stick model with color-coding, atomic information (electrostatic fields, valence number, atom name), and other relevant information.

[12] MIMAS; http://www.mimas.ac.uk/crossfire/
[13] MDL; http://www.mdl.com/
[14] PubChem; http://pubchem.ncbi.nlm.nih.gov/

References

BÖTSCHI, K. 2005. Evaluation of Augmented Chemistry, a Tangible User Interface for high-school organic chemistry education. ETH E-Collection, M.Sc. dissertation. http://e-collection.ethbib.ethz.ch/show?type=dipl&nr=171

FJELD, M., VOEGTLI, B. 2002. Augmented Chemistry: An Interactive Educational Workbench. *Video program and proc. of International Symposium on Mixed and Augmented Reality (ISMAR),* 2002, pp. 259-260.

FJELD, M., HOBI, D., WINTERTHALER, L., VOEGTLI, B., JUCHLI, P. 2004. Teaching electronegativity and dipole moment in a TUI. Proc. *Advanced Learning Technologies*, IEEE, 2004, pp. 792-794.

GILLET, A., SANNER, M., STOFFLER, D., OLSON, A. 2005. Tangible Augmented Interfaces for Structural Molecular Biology, *IEEE Computer Graphics and Applications*, vol. 25, no. 2, pp. 13-17.

ISHII, H., UNDERKOFFLER, J., CHAK, D., PIPER, B., BEN-JOSEPH, E., YEUNG, L., KANJI, Z. 2002. Augmented Urban Planning Workbench: Overlaying Drawings, Physical Models and Digital Simulation, in *Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR '02)*, p. 203.

KATO, H., BILLINGHURST, M., POUPYREV, I., IMAMOTO, K., TACHIBANA, K. 2000. Virtual Object Manipulation on a Table-Top AR Environment. *Proc. Int. Symposium on Augmented Reality (ISAR)*, 2000, pp. 111-119.

SHON, Y., MCMAINS, S. 2004. Evaluation of drawing on 3D surfaces with haptics, *Computer Graphics and Applications,* IEEE 2004, pp. 40-50.

VAN LIERE, R., KOK, A., MARTENS, J.-B., VAN TIENEN, M. 2005. Interacting with Molecular Structures: User Performance versus System Complexity. *Proc. EGVE 2005*.

VOEGTLI, B. 2002. *Augmented Collaboration*. Thesis by Benedikt Voegtli. HyperWerk FHBB, March 2002. http://hyperwerk.ch/admin/pdf_content/augmented.cemistry_beni.v.pdf

Acknowledgements

Benedikt Voegtli and Patrick Juchli designed and developed the AC system at HyperWerk FHBB in 2001/2002; their advisors/coaches were Morten Fjeld and Regine Halter. Irena Kulka and Céline Studer helped develop the educational content of AC. Jan Kulka and Hanspeter Huber provided assistance in the field of chemistry education. Edouard Bannwart helped design the AC system setup. Bruno Paneth of aprentas proved a highly competent discussion partner for chemistry education issues and made a substantial contribution to keeping the AC project on a productive track. Erik Tobin was of great help in putting this paper in the appropriate scientific form.

Automatic Conversion of Traffic Accident Reports into 3D Animations

Richard Johansson*and Pierre Nugues †
Department of Computer Science
Lund University, Sweden

Abstract

This paper describes a system that automatically converts narratives into 3D scenes. The texts, written in Swedish, describe road accidents. One of the key features of the program is that it animates the generated scene using temporal relations between the events. We believe that this system is the first text-to-scene converter that is not restricted to invented narratives.

The system consists of three modules: natural language interpretation based on information extraction (IE) methods, a planning module that produces a geometric description of the accident, and finally a visualization module that uses Java3D to render the geometric description as animated graphics.

We performed a small user study to evaluate the quality of the visualization. The results validate our choice of methods, and since this is the first evaluation of a text-to-scene conversion system, they also provide a baseline for further studies.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Animations; I.2.7 [Artificial Intelligence]: Natural Language Processing—Text Analysis; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search—Plan execution, formation, and generation

Keywords: text-to-scene conversion, 3D graphics, natural language processing

1 Introduction

Automatic text-to-scene conversion consists in synthesizing a 2D or 3D geometric description from a text and in displaying it. The scene can be static or animated. Animated 3D graphics have some advantages for the visualization of information. They can reproduce a real scene more accurately and render a sequence of events. In addition, when 3D graphics are coupled with virtual environments, they enable users to navigate in a 3D world and interact with objects.

The conversion of natural language texts into graphics has been investigated in a few projects. NALIG [Adorni et al. 1984] is an early example of them that was aimed at recreating static 2D scenes. One of the major goals of the project was to study relationships between space and prepositions. NALIG considered simple phrases in Italian of the type subject, preposition, object that in spite of their simplicity can have ambiguous interpretations. From what is described in the papers, NALIG has not been extended to process sentences and even less to texts. WordsEye [Coyne and Sproat 2001] is a more recent system that recreates 3D animated scenes from short descriptions. The interpretation of a narrative is based on semantic frames and a good deal of inferences about the environment. WordsEye does not address real world stories. The narratives cited as examples resemble imaginary fairy tales. In addition, all the cited texts appear to have been invented by the authors and not collected from kids, for instance. CogViSys is a last example that started with the idea of generating texts from a sequence of video images. The authors found that it could also be useful to reverse the process and generate synthetic video sequences from texts. The system is limited to the visualization of single vehicle maneuvers at an intersection as the one described in this two-sentence narrative: *A car came from Kriegstrasse. It turned left at the intersection* [Arens et al. 2002]. The authors give no further details on the results.

2 Overview of the Carsim System

The Carsim[1] system [Johansson et al. 2005; Dupuy et al. 2001] is a text-to-scene converter that handles real texts and that we evaluated using quantitative methods. The program generates 3D graphics from traffic accident reports generally collected from web sites of Swedish newspapers. One of its key features is that it takes time and temporal relations between events into account to animate the synthesized scene.

Narratives of a car accidents often make use of space descriptions, movements, and directions that are sometimes difficult to grasp for readers. We believe that forming consistent mental images is necessary to understand them properly. However, some people have difficulties in imagining situations and may need visual aids predesigned by professional analysts.

Carsim tries to address this need. It is intended to be a helpful tool that can enable people to imagine a traffic situation and understand the course of events properly. To generate a 3D scene, Carsim combines natural language processing components and a visualizer. The language processing module adopts an information extraction (IE) strategy and includes machine learning methods to solve coreference, classify predicate/argument structures, and order events temporally.

However, as real texts suffer from underspecification and rarely contain a detailed geometric description of the actions, information extraction alone is insufficient to convert narratives into images automatically. To handle this, Carsim infers implicit information about the environment and the involved entities from key phrases in the text, knowledge about typical traffic situations, and properties of the involved entities. The program uses a visualization planner that applies spatial and temporal reasoning to find the simplest configuration that fits the description.

*e-mail: richard@cs.lth.se
†e-mail:pierre@cs.lth.se

[1]An online demonstration of the system is available at `http://www.lucas.lth.se/lt`.

2.1 A Corpus of Traffic Accident Descriptions in Swedish

Carsim has been developed using a corpus of reports written in Swedish. As development set, we collected approximately 200 reports of road accidents from various newspapers. The task of analyzing the news reports is made more complex by their variability in style and length. The amount of details is overwhelming in some reports, while in others most of the information is implicit. The complexity of the accidents described ranges from simple accidents with only one vehicle to multiple collisions with several participating vehicles.

Although our work has concentrated on the press clippings, we also have a small set of accident reports extracted from the STRADA database (Swedish TRaffic Accident Data Acquisition) of Vägverket, the Swedish traffic authority.

The next text is an excerpt from our test corpus. This report is an example of a press wire describing an accident.

> *Tre personer omkom när en buss och personbil på måndagen kroc*k*ade på väg 55 vid Fornebo i närheten av Flen. Det var ett barn och två vuxna som färdades i personbilen som omkom i olyckan. Ytterligare ett barn, en flicka, fanns i bilen, men kunde ta sig ut. - Hon fick hjälp av en person att ta sig ut ur bilen, berättar Mats Elfwén, räddningsledare vid räddningstjänsten i Flen, för TT. Han vet inte hur olyckan gick till. - Av någon anledning kom personbilen över på fel sida med sladd. Bussföraren försökte undvika den, men det blev en frontalkollision, säger Mats Elfwén. Vid krocken fattade personbilen eld. Flickan som räddades ur bilen fördes till sjukhus med bland annat brännskador. Ungefär 15 personer från räddningstjänsterna i Flen och Malmköping deltog i arbetet vid olyckan.*
>
> Svenska dagbladet, November 11, 2002
>
> Three persons died when a bus and a passenger car collided on Monday on road 55 near Fornebo in the vicinity of Flen. A child and two adults, who traveled in the passenger car, died in the accident. Another child, a girl, was in the car but was able to escape. -She was assisted by a person to get out of the car, reported Mats Elfwén, head of the rescue team in Flen, to the Swedish News Agency. He does not know how the accident occurred. For some reason, the passenger car slid and came over on the wrong side of the road. The bus driver tried to avoid it, but ended in a frontal collision, said Mats Elfwén. During the collision, the passenger car caught fire. The girl that could escape the car was sent to hospital with burns. Approximately 15 persons from the rescue team in Flen and Malmköping participated in the rescue.
>
> The text above, our translation.

2.2 Carsim's Architecture

Carsim's architecture consists of a pipeline of three modules where each module carries out one step of the conversion process (see Figure 1).

- A *natural language processing* module interprets the text to fill a template – an intermediate symbolic representation.
- A *spatio-temporal planning and inference* module produces a full geometric description given the symbolic representation.
- A *graphical* module renders the geometric description as graphics using the Java3D library.

Carsim's language processing module uses information extraction techniques. It reduces the text content to a tabular structure – the template – that outlines what happened. We use this template as an intermediate representation between texts and geometry. This is made necessary because the information expressed by most reports usually has little affinity with a geometric description. Exact and explicit accounts of the world and its physical properties are rarely present. In addition, our vocabulary is finite and discrete, while the set of geometric descriptions is infinite and continuous.

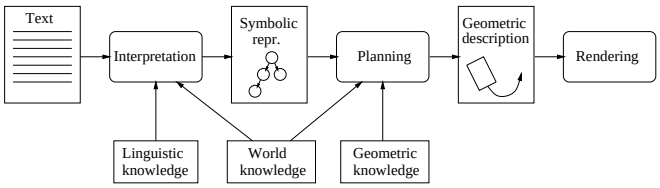


Figure 1: System architecture.

Once the NLP module has interpreted and converted a text, the planner maps the resulting symbolic representation of the world, the entities, and behaviors, onto a complete and unambiguous geometric description in a Euclidean space.

Certain facts are never explicitly stated, but are assumed by the author to be known to the reader. This includes linguistic knowledge, world knowledge (such as traffic regulations and typical behaviors), and geometric knowledge (such as typical sizes of vehicles). The language processing and planning modules take this knowledge into account in order to produce a credible geometric description that can be visualized by the renderer.

2.3 Knowledge Representation

The knowledge representation contained in the template has to manage the following trade-off. In order to be able to describe a scene, it must contain enough information to make it feasible to produce a consistent geometric description, acceptable to the user. On the other hand, the representation has to be close to ways human beings describe things to capture information in the texts.

We used four concept categories that we ordered in an inheritance hierarchy. Each category is implemented as predefined attribute/values slots:

- *Objects*. These are typically the physical entities that are mentioned in the text, but we might also need to present abstract entities as symbols in the scene. Each object has a type that is selected from a predefined, finite set. `Car` and `Bus` are examples of object types.
- *Events*. They correspond intuitively to an activity that goes on during a period in time and here to the possible object behaviors. We represent events as entities with a type from a predefined set. `Impact` and `CatchFire` are examples.
- *Relations and Quantities*. The objects and the events need to be described and related to each other. The most obvious examples of such information are *spatial* information about objects and *temporal* information about events. We should be able to express not only exact quantities, but also qualitative information (by which we mean that only certain fundamental distinctions are made). `Behind`, `FromLeft`, and `During` are examples of spatial and temporal relations.
- *Environment*. The environment of the accident is important for the visualization to be understandable. Significant environmental parameters include light, weather, road conditions,

and type of environment (such as rural or urban). Another important parameter is topography, but we have set it aside since we have no convenient way to express this qualitatively.

2.4 Natural Language Processing

The natural language processing module uses information extraction techniques. It consists of a sequence of components (Figure 2). The first components carry out a shallow parse: part-of-speech tagging, noun phrase chunking, complex word recognition, and clause segmentation. This is followed by a cascade of semantic markup components: named entity recognition, temporal expression detection, object markup and coreference, and predicate argument detection. A temporal component uses verb tenses and other information to infer the temporal structure of the course of events. Finally, the marked-up structures are interpreted, which results in a symbolic representation of the accident.

The development of the IE module has been made more complex by the fact that few tools or annotated corpora are available for Swedish. The only significant external tool we have used is the Granska part-of-speech tagger [Carlberger and Kann 1999].

3 The Planning and Graphical Modules

We use a planner to create the animation from the information extracted by the natural language processing module. It first determines a set of constraints that the animation needs to fulfill. Then, it goes on to find the initial directions and positions. Finally, it uses a search algorithm to find the trajectory layout. This separation into steps does not allow backtracking and introduces a risk of bad choices. However, it reduces the computation load and proved sufficient for the texts we considered, enabling an interactive generation of 3D scenes and a better user experience.

3.1 Setting Up the Constraints

The constraints on the animation are created using the detected events and the spatial and temporal relations combined with the implicit knowledge about the world. The events are expressed as conjunctions of primitive predicates about the objects and their behavior in time. For example, if we state that there is an `Overtake` event where O_1 overtakes O_2, this is translated into the following proposition:

$$\exists t_1, t_2 . \mathit{MovesSideways}(O_1, \mathit{Left}, t_1) \\ \wedge \mathit{Passes}(O_1, O_2, t_2) \wedge t_1 < t_2$$

In addition, other constraints are implied by the events and our knowledge of the world. For example, if O_1 overtakes O_2, we add the constraints that O_1 is initially positioned behind O_2, and that O_1 has the same initial direction as O_2. Other constraints are added due to the non-presence of events, such as

$$\mathit{NoCollide}(O_1, O_2) \equiv \neg \exists t . \mathit{Collides}(O_1, O_2, t)$$

if there is no mentioned collision between O_1 and O_2.

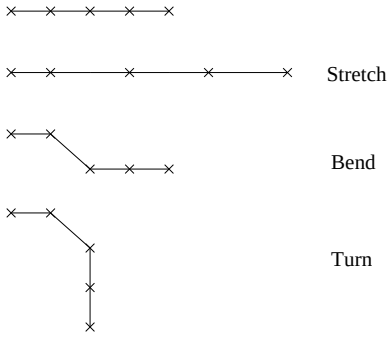


Figure 3: The elementary movements.

3.2 Finding Initial Directions and Positions

We use constraint propagation techniques to infer initial directions and positions for all the involved objects. We first set those directions and positions that are stated explicitly. Each time a direction is uniquely determined, it is set and this change propagates to the sets of available choices of directions for other objects, whose directions have been stated in relation to the first one. When the direction can't be determined uniquely for any object, we pick one object and set its direction. This goes on until the initial directions have been inferred for all objects.

3.3 Finding the Trajectories

After the constraints have been set up, we use the IDA* search method to find a trajectory layout that is as simple as possible while violating no constraints. The trajectories are initially straight, and are modified incrementally until a solution is found. The three types of modifications to the trajectories (i.e. the elementary movements the vehicles can make) are shown in Figure 3. As a heuristic function to guide the search, we use the number of violated constraints multiplied by a scaling constant in order to keep the heuristic admissible.

The most complicated accident in our development corpus contains 8 events, which results in 15 constraints during search, and needs 6 modifications of the trajectories to arrive at a trajectory layout that violates no constraints. This solution is found in a few seconds. Most accidents can be described using only a few constraints.

At times, no solution is found within reasonable time. This typically happens when the IE module has produced incorrect results. In this case, the planner backs off. First, it relaxes some of the temporal constraints (for example: *Simultaneous* constraints are replaced by *NearTime*). Next, all temporal constraints are removed.

3.4 A Planning Example

To illustrate the planning problem, we give an example of a common kind of traffic accident: *A* overtakes *B*, forcing *C* (coming from the opposite direction) off the road. We formalize this using the following constraints:

- $\exists t_1 \mathit{MovesSideways}(A, \mathit{Left}, t_1)$
- $\exists t_2 \mathit{Passes}(A, B, t_2)$
- $\exists t_3 \mathit{LeavesRoad}(C, t_3)$

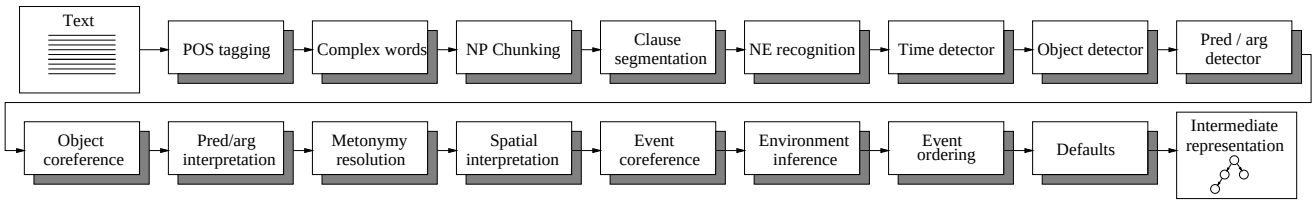


Figure 2: Architecture of the language interpretation module.

- $t_2 > t_1$
- $t_3 > t_2$
- $NoCollide(A,B)$
- $NoCollide(A,C)$
- $NoCollide(B,C)$

We start from initial trajectories, and the search procedure modifies them incrementally. A minimal solution is $Bend(A)$, $Stretch(A)$, $Bend(C)$, meaning that vehicles A and C perform sideways movements, and that vehicle A accelerates (while overtaking).

3.5 Rendering the Geometric Description

We use the Java3D library to render the geometric description as 3D graphics. Java3D is a more or less platform-independent 3D graphics library on top of platform-dependent low-level libraries like OpenGL or DirectX.

Most objects are described using VRML models, although some are implemented using procedural techniques (fires, for example).

Figures 4 and 5 show an example corresponding to the text from Subsection 2.1.

4 Evaluation of the Visualization

To evaluate the system, we used 50 previously unseen texts, which had been collected from newspaper sources on the web. The size of the texts ranged from 36 to 541 tokens. Four users were shown the animations of subsets of the 50 test texts.

The users graded the quality of animations using the following scores: 0 for wrong, 1 for "more or less" correct, and 2 for perfect. The average score was 0.91. The number of texts that had an average score of 2 was 14 (28 percent), and the number of texts with an average score of at least 1 was 28 (56 percent). These figures demonstrate that the chosen strategy is viable, especially in a restricted context like the traffic accident domain. However, interpretation of the figures is difficult since there are no previously published results. In any case, they provide a baseline for further studies, possibly in another domain.

To determine whether the small size of our test group introduced a risk of invalid results, we calculated the standard deviation of annotations[2], and we obtained the value of 0.45. Replacing all annotations with random values from the same probability distribution resulted in a standard deviation of 0.83 on average. In addition, the pairwise correlation of the annotations is 0.75. This suggests that the agreement among annotators is enough for the figures to be relevant.

During discussions with users, we had a number of unexpected opinions about the visualizations. One important example of this is the implicit information they infer from reading the texts. For example, given a short description of a crash in an urban environment, one user imagined a collision of two moving vehicles at an intersection, while another user interpreted it as a collision between a moving and a parked car.

This user response shows that the task of imagining a situation is difficult for humans as well as for machines. Furthermore, while some users have suggested that we improve the realism (for example, the physical behavior of the objects), discussions generally made it clear that the semi-realistic graphics that we use (see Figures 4 and 5) may suggest to the user that the system knows more than it actually does. Since the system visualizes symbolic information, it may actually be more appropriate to present the graphics in a more "abstract" manner that reflects this better, for example via symbolic signs in the scene.

5 Conclusion and Perspectives

We have presented system based on information extraction techniques and symbolic visualization that converts real texts into 3D scenes. It creates animated graphics by taking into account temporal relations between events mentioned in a text and using a planner.

We have provided a quantitative evaluation of a text-to-scene conversion system, which shows promising results that validate our choice of methods and set a baseline for future improvements. As far as we know, Carsim is the only text-to-scene conversion system that has been developed and evaluated using noninvented narratives.

In the future, we would like to extend the system to deeper levels of semantic processing. While the current prototype uses no external knowledge, we would like to integrate additional knowledge sources in order to make the visualization more realistic. An important example of this is geographical and vehicle information, which could be helpful in improving the realism and in creating a more accurate reconstruction of the environment and animation. Another topic we would like to address would be to merge a set of narratives describing a same accident into a unique 3D scene as the animations manually produced by the National Transportation Safety Board (NTSB) in the United States[3].

[2] We calculated this using the formula $\sqrt{\frac{\sum(x_{ij}-\dot{x}_i)^2}{\sum(n_i-1)}}$, where x_{ij} is the score assigned by annotator j on text i, $\dot{x}_i$ the average score on text i, and n_i the number of annotators on text i.

[3] See for instance www.ntsb.gov/events/2000/central_bridge/cb_video.htm

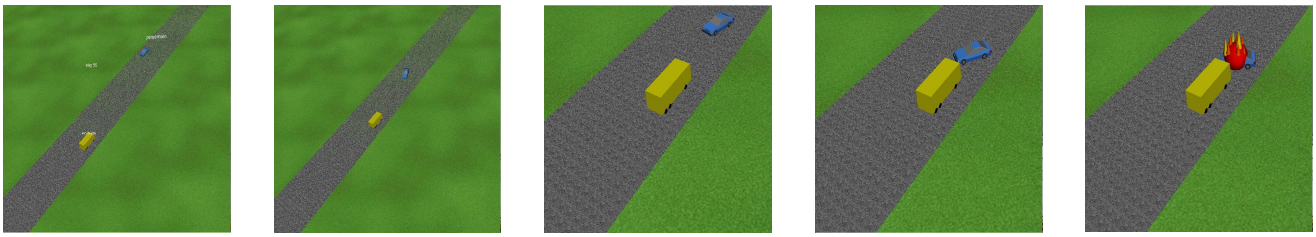


Figure 4: Screenshots from the animation of the text above.

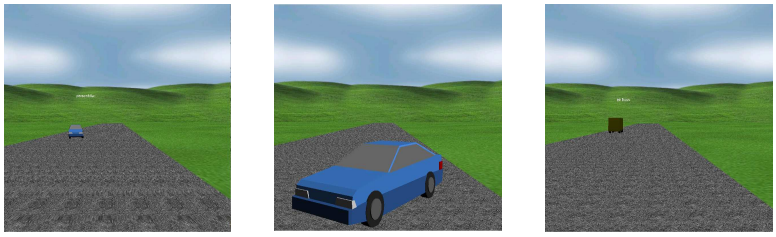


Figure 5: Points of view from the bus and the car.

Acknowledgements

This work was partly supported by grant number 2002-02380 from the Språkteknologi program of Vinnova, the Swedish Agency of Innovation Systems.

References

ADORNI, G., MANZO, M. D., AND GIUNCHIGLIA, F. 1984. Natural language driven image generation. In *Proceedings of COLING 84*, 495–500.

ARENS, M., OTTLIK, A., AND NAGEL, H.-H. 2002. Natural language texts for a cognitive vision system. In *ECAI2002, Proceedings of the 15th European Conference on Artificial Intelligence*, F. van Harmelen, Ed.

CARLBERGER, J., AND KANN, V. 1999. Implementing an efficient part-of-speech tagger. *Software Practice and Experience 29*, 815–832.

COYNE, B., AND SPROAT, R. 2001. WordsEye: An automatic text-to-scene conversion system. In *Proceedings of the Siggraph Conference*.

DUPUY, S., EGGES, A., LEGENDRE, V., AND NUGUES, P. 2001. Generating a 3D simulation of a car accident from a written description in natural language: The Carsim system. In *ACL2001: Workshop on Temporal and Spatial Information Processing*, 1–8.

JOHANSSON, R., BERGLUND, A., DANIELSSON, M., AND NUGUES, P. 2005. Automatic text-to-scene conversion in the traffic accident domain. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 1073–1078.

Visualisation as a tool for understanding fibre network behavior

Jonas Lindemann*
Lunarc
Lund University

Gran Sandberg†
Division of Structural Mechanics
Lund University

Ola Dahlblom‡
Division of Structural Mechanics
Lund University

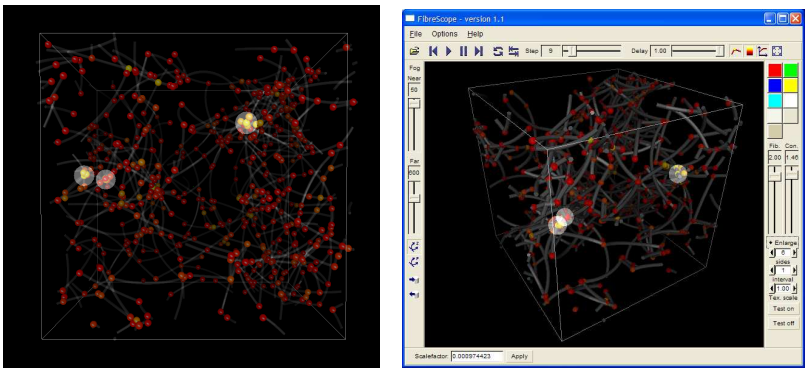


Figure 1: Connection breakage visualisation.

Abstract

Fibre networks are material with a structure consisting of fibres. At Structural Mechanics simulation methods for studying the behaviour of such networks has been developed. Because the behaviour of these materials are depends on three-dimensional interaction between fibres in the network it can be difficult to interpret the results without good visualisation tools and methods. The standard tools available are often designed with other materials and structures. To enable effective visualisation of these simulations custom tools and methods has been developed in parallel with the development of the simulation code, providing tools for better understanding the behavior of the simulations and to compare with real experiments.

Keywords: fibre networks, real-time, visualisation, finite element method

1 Introduction

Fibre network materials are materials with a structure consisting of fibres. Examples of such materials can be found in insulation material, diapers and paper. Structural Mechanics has during several years developed methods for studying the behaviour of different fibre network materials [Heyden 2000]. Visualising the results from the simulations been difficult using commonly found visualization software. This paper illustrates how visualisation can be used as an integral part in the development process of a new simulation code, enabling better understanding of simulation results as well as evaluating initial network configurations compared to real materials.

In earlier work [Lindemann and Dahlblom 2002], a texture based method was used to reduce the geometry complexity, enabling larger fibre networks to be visualised in real-time. This method is integrated in a fibre network post-processor, FibreScope.

*e-mail: jonas.lindemann@byggmek.lth.se
†e-mail:goran.sandberg@byggmek.lth.se
‡e-mail:ola.dahlblom@byggmek.lth.se

2 Evaluation of simulation results

Results from a 3D fibre network simulation consist of ”snapshots” of the fibre network at different time steps in the calculation. In addition to this information on each fibre-to-fibre connection is also stored, so that fibre breakage can be studied. A simulated fibre network can consist of several thousand fibres and connection points. To evaluate the simulation results the visualisation tool must be able to visualise the deformation history of each fibre as well as highlighting connection point usage and breakage.

The common method of visualising fibres is to sweep a cross section over a spine (extrusion). This method is good when detail is needed, but generates a lot of geometry when the networks are large.

2.1 Visualisation of fibre structure and deformation

When the networks become larger, the extrusion based method become more and more costly in terms of real-time performance. To overcome this, a texture based method was developed and implemented in earlier work [Lindemann and Dahlblom 2002], reducing the geometry demands when visualising large networks.

The method renders the fibre as a simple band consisting of view-aligned quads (billboarding). A gradient texture is applied on the band giving it the illusion of a rounded fibre. Using this technique much larger networks can be visualised using similar hardware. Figure 2 shows the extrusion based fibre compared to the banded extruded fibre.

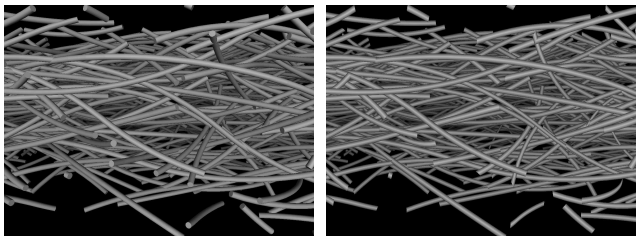


Figure 2: Fibres rendered with the extrusion based (left) and the texture based method (right)

2.2 Visualising connection point usage and breakage

An essential part in the simulation model is the connection point between fibres in the network. When the network is subjected to loading, these connection points will become stressed and eventually break. The simulation model [Heyden 2000] provides information about connection point usage. To be able to study how the behaviour of fibre network it important that the connection point usage is visualised in an intuitive way.

In a real fibre network, the connection point is very small, so it is not possible visualise the actual connection point. In the previous work [Lindemann and Dahlblom 2002] colored spheres where used to indicate the connection points. Using a colour scale the sphere is coloured to indicate the connection point usage. When the connection point usage is over a certain value, it will break. To indicate breakage the sphere is scaled to produce a ”popping” effect, to make the user aware of breaking connection points. When evaluating this method proved less effective, because the scaled spheres where obscured by the fibre network and other spheres.

To enable the user to be aware of connection point breakage in the entire fibre network a different approach is taken. Instead of scaling the connection point representation an additional ”highlight sphere” is rendered on top of the connection point. To be able to see the highlight sphere even if the connection point is obscured by other spheres and geometry, it is rendered without depth buffer turned on (`glDisable(GL_DEPTH_BUFFER)`). To retain the visibility of the actual connection point indicator the sphere is also rendered using additive blending (`glBlendFunc(GL_ONE,GL_ONE)`).

Because breakage occurs at discrete moment in the simulation, and the behavior just before and after the breakage is important the developed highlighting method highlights the connection points when the exceed 95% usage or a user defined value. This enables the user to zoom-in on interesting events and see the breakage occuring, as seen in Figure 3.

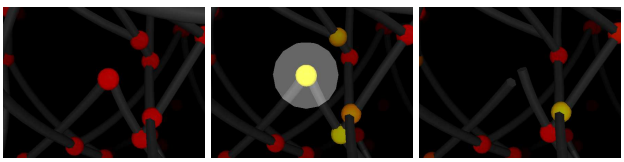


Figure 3: High usage, enlarged sphere, after break

Using this techniques the highlight for the connection point breakage will be visible at all viewpoints even if the real connection point is obscured by other geometry, as seen in Figure 4.

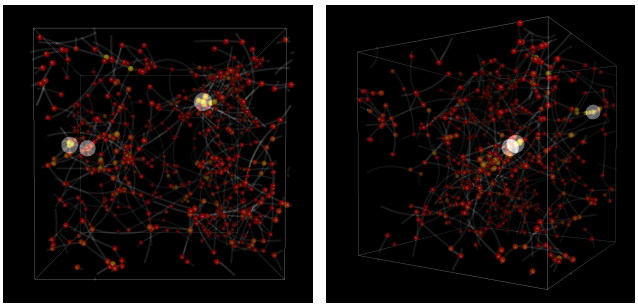


Figure 4: 3 connection breaks viewed from different viewpoints.

2.3 The tool: FibreScope

The methods described in the previous sections are all implemented in the post processing application FibreScope, see Figure 5, which initial development was done in previous work [Lindemann and Dahlblom 2002].

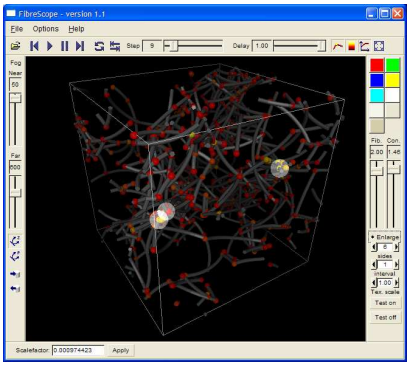


Figure 5: FibreScope post processor user interface

The main design concept of FibreScope is to implement a ”virtual” microscope that can be used to analyse the simulation results produced with the simulation code. The user can easily browse and animate the simulation time steps and in the same time rotate, pan and zoom. The user interface of is also designed to provide as much control as possible for the user, directly in the main window, eliminating the need for dialog windows. The user is encouraged to experiment with the different parameters for the visualization methods implemented. FibreScope has also been extended, so that active stereo equipment can be used, providing true depth perception.

FibreScope is a platform independent C++ application that can be run on Microsoft Windows, Mac OS X and Linux. 3D rendering is implemented using Ivf++ [Lindemann] a thin object-oriented library on top of OpenGL [Ope 2005]. The User interface is implemented using the Fast Light Toolkit (FLTK) [Spitzak 2005], to provide the necessary platform independence.

3 Evaluation of generated networks

Initially the focus of this work was on visualising the results from fibre network simulations. Later on it was clear that the visualisation tools could also provide valuable feedback in the process of generating the initial fibre network configurations. That is to generate fibre networks resembling real fibre networks. A master thesis

work [Edlind 2003] studied methods for generating fibre networks with specific properties, such as certain distribution of fibre orientations or fibre curvatures.

To compare the generated fibre networks with images of real fibre networks, 3D Studio MAX and FibreScope was used. FibreScope was used to quickly view a generated network, 3D Studio MAX was used to generate high resolution images for direct comparison with real images. Figure 6 shows a real fibre material compared to one generated material.

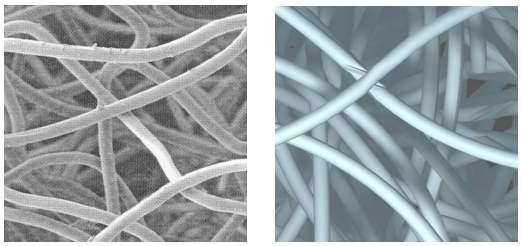


Figure 6: Real fibre material (left), Generated material rendered with 3D Studio MAX (right)

Because these generated networks have different properties in different directions, a special mode in FibreScope has been added to enable the user to study the depth complexity. This mode is implemented using a standard technique employed in many other visualisation applications. The basic idea is to render all objects with the depth buffer disabled and blending all objects using an adding function. By specifying colour components in a special way (red=0.2, green=0.1, blue=0.0), a colour scale will be produced where dark red represent low depth complexity and yellow colour represents high depth complexity. The colours must be calibrated for a specific depth complexity. Figure 7 shows a network with low complexity and figure 8 the same network from another angle now having a higher depth complexity.

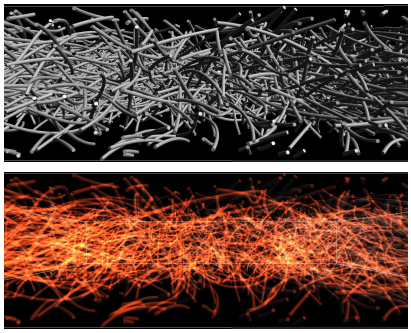


Figure 7: Network viewed in a direction having low depth complexity

To aid in the process of developing the generation methods FibreScope has been enhanced with routines for reading these generated networks. A full screen viewer was also implemented that could be used with the visualisation equipment and the Reflex RealityCenter at Lund University.

4 Conclusion

Fibre network simulations can consist of several thousands of fibres with snapshots of the simulation state stored at each time step. This

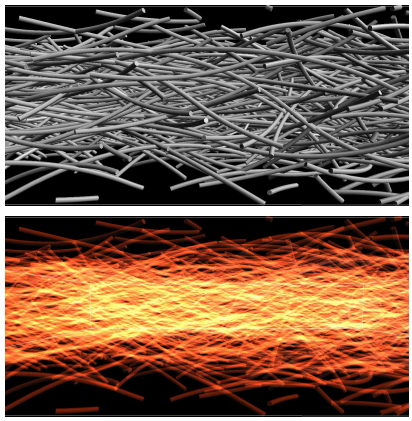


Figure 8: Network viewed in a direction having high depth complexity

produces a lot of data to analyse. Effective visualisation methods and tools are very important in this task. To aid in this task a banded texture approach was developed in previous work [Lindemann and Dahlblom 2002], to aid in the visualisation of large fibre networks.

Another important aspect in fibre network simulations is the connection point between fibres and how these break during load. This paper describes a enchanced method for highlighting connection point breakage, enabling the user to see connection point breakage even if the connection point is obscured by other geometry.

The tools developed has also been used to study methods for generating the initial fibre geometry, providing a rapid visualisation before more photo-realistic renderings are done in more advanced rendering packages, such as 3D Studio MAX.

This paper gives an overview how visualisation can be used as an integrated tool in the process of developing a simulation code. During the development of the simulation code developed by [Heyden 2000], tools for fibre network visualisation has been developed in parallel, providing important information about the behavior of the simulated networks. The FibreScope application is continually developed to integrate the developed methods into a easy to use user interface enabling the user to experiment with many parameters to achieve the desired results or findings.

References

EDLIND, N. 2003. *Modelling and Visualization of the Geometry of Fibre Materials*. Master's thesis.

HEYDEN, S. 2000. *A 3D Network Model for Evaluation of Mechanical Properties of Cellulose Fibre Fluff*. PhD thesis.

LINDEMANN, J. *Interactive Visualisation Framework - User's guide*. Division of Structural Mechanics, Lund University.

LINDEMANN, J., AND DAHLBLOM, O. 2002. Real-time visualisation of fibre networks. *The Visual Computer 18*, 20–28.

2005. *OpenGL – The Industry's Foundation for High Performance Graphics*. http://www.opengl.org.

SPITZAK, B. 2005. *Fast Light Toolkit (FLTK)*. http://www.fltk.org.

Augmented Reality on Mobile Phones - Experiments and Applications

Anders Henrysson*
NVIS
Linkoping University, Sweden

Mark Billinghurst†
HITLab NZ
University of Canterbury, New Zealand

Mark Ollila‡
NVIS
Linkoping University, Sweden

Figure 1: Three examples of Augmented Reality applications running on a mobile phone. The left image shows visualization of a static object. The center image shows a face-to-face collaborative game. The right image shows an object manipulation and scene assembly application.

Abstract

Mobile phones have reached a level where it is possible to run self-contained Augmented Reality applications using the built-in camera for optical tracking. In this paper we present some of our work in this area. We have created a custom port of the ARToolKit library to the Symbian mobile phone operating system and then developed sample applications which have been evaluated. These include a face-to-face collaborative AR game where we conducted a user study to evaluate multi-modal feedback. We also examined user interface issues where an AR enabled mobile phone acts as an interaction device. Additionally, we discuss how traditional 3D manipulation techniques apply to this new platform. We also describe a mobile phone based Augmented Reality application for 3D scene assembly, which adds a 6 DOF isomorphic interaction technique for manipulating 3D content.

1 Introduction

With the integration of cameras and full color displays, mobile phones have developed into an ideal platform for Augmented Reality (AR). Now that it is technically possible, it is important to conduct research on the types of AR applications that are ideally suited to mobile phones and user interface guidelines for developing these applications. This is significant because the widespread adoption of mobile phones means that this platform could be one of the dominant platforms for AR applications in the near future.

*e-mail: andhe@itn.liu.se
†e-mail:mark.billinghurst@canterbury.ac.nz
‡e-mail:marol@itn.liu.se

AR is a technology that allows a user to see virtual imagery overlaid and registered with the real world. Traditionally the AR content was viewed through a head mounted display (HMD). Wearing a HMD leaves the users hands free to interact with the virtual content, either directly or using an input device such as a mouse or digital glove.

For handheld and mobile phone based AR the user looks through the screen of the device to view the AR scene and needs at least one hand to hold the device. The user interface for these applications is very different than those for HMD based AR applications. Thus there is a need to conduct research on interaction techniques for handheld AR displays, and to produce formal user studies to evaluate these techniques.

New opportunities in mobile phone interaction have emerged with the integration of cameras into the phones. By analyzing the video stream captured by the camera, using simple image processing on the phone, it is possible to estimate the movement of the device. Such estimation is the essential component in a Augmented Reality setup.

In this paper we present the implementation of several possible manipulation techniques and the results of a user study conducted to identify which of these techniques is the most usable. These techniques can be used to provide a 6 DOF interface. We show how different strategies can be combined for manipulation of a general 3D scene using a standard mobile phone. We describe the first example of using phone motion to manipulate graphical objects in 6 DOF to create virtual scenes.

Another interesting area for mobile phone based AR is for supporting collaborative AR applications. Mobile phones are already designed to support local and remote communication and so provide a natural platform for collaborative AR. For example a Bluetooth enabled mobile phone can be used for face-to-face gaming or messaging, while the cellular network supports voice and video calls.

In the next section we review related work in the area of mobile AR, collaborative AR and virtual object manipulation on a handheld platform. Next we talk about user interface aspects of mobile phone AR and the software platform we have developed to support phone based AR applications. We then describe our work in different areas of mobile phone AR, especially virtual object manip-

ulation, scene assembly and collaborative AR. Finally we provide some directions for future research.

2 Related Work

Our work draws on a rich legacy of previous work in handheld augmented reality, collaborative augmented reality, AR interaction techniques and mobile phone gaming.

The first mobile AR set-ups such as Feiners Touring Machine [1997] featured a backpack computer and an HMD. From these days it was obvious that what was carried in a backpack would one day be held in the palm of the hand. Unlike the backpack systems, handheld collaborative AR interfaces are unencumbering and ideal for lightweight social interactions.

Rekimotos Transvision system explored how a tethered handheld display could provide shared object viewing in an AR setting [1996]. Transvision consists of a small LCD display and a camera. These are connected by a cable to a computer that performs the augmentation. Two users sit across the table and see shared AR content shown on the displays. They can select objects by ray casting and once selected objects are fixed related to the LCD and can be moved. The AR-PAD interface [Mogilev et al. 2002] is similar, but it adds a handheld controller to the LCD panel. AR-PAD decouples translation and rotation. A selected object is fixed in space relative to the LCD panel and can be moved by moving the panel. Rotation is performed using a trackball input device. These custom configurations show that if the AR display is handheld the orientation and position of the display can be used as an important interaction tool.

The first commercially available handheld platform to be used for AR applications was the PDA. First there was work such as the AR-PDA project [2001] in which the PDA was used as a thin client for showing AR content generated on a remote server. This was necessary as the early PDAs did not have enough capability for stand-alone AR applications. Then in 2003 Wagner [2003] ported ARToolKit [ART] to the PocketPC and developed the first self contained PDA AR application. Handheld AR applications such as the Invisible Train [Wagner et al. 2005] also show an interesting combination of interacting with the AR content by interacting in the world and with the device itself.

Mobile phone based AR has followed a similar development path. Early phones did not have enough processing power so researchers also explored thin client approaches. For example, the AR-Phone project [Cutting et al. 2003] used Bluetooth to send phone camera images to a remote sever for processing and graphics overlay, taking several seconds per image. However, Henrysson recently ported ARToolKit over to the Symbian phone platform [2004], while Moehring developed an alternative custom computer vision and tracking library [2004]. This work enables simple AR applications to be developed which run at 7-14 frames per second, but requires a 3D marker.

By visually tracking real objects, the camera phone can be used for 6 DOF input. Hachet [2005] has developed a 3 DOF bimanual camera based interface for interaction both on the device itself and for using a PDA as a 3D mouse. The approach is similar to ours in that it establishes the position and orientation of the device by analyzing the video stream captured by the camera. Rohs Visual Codes [2004] is an example of mobile phone barcode reading. By recognizing and tracking a pattern, the phone movements can be estimated and used as input. The pattern can also be associated with phone functions and act as a menu item. Hansens Mixed Interaction Spaces [2005] uses a similar approach by tracking a circle. Non of these works have proven to be sufficient for 3D AR applications.

Finally, work in mobile phone gaming has been used to inform our AR application design. There are several examples of 3D graphics applications on mobile phones. The vast majority are games that provide joystick type control of vehicles and objects in 3D environments. Larsen [2002] describe one of the first 3D applications for the mobile phone with more complex object manipulation. This is a client server setup where the rendering of the bricks is made on the server in addition to collision detection. There is no mentioning of interactive change of the view. Transformation is restricted to 2D translation. Although there are thousands of games available for mobile phones, there is only a handful that use camera input. Two of the best known are Mosquito Hunt and Marble Revolution. Neither of these games are collaborative or true AR applications, but they do show that camera and phone motion can be used to create compelling game experiences.

The application most related to our work in collaborative AR is Hakkarainens Symball game [2005]. This is a two person collaborative table tennis game which uses camera phones that are Bluetooth equipped. The user control a virtual paddle by moving the phone relative to a colour that is tracked. Once again this is not a true AR experience, but it is the first example of a compelling collaborative game on phone that user camera input.

3 Interaction

There have been several interface metaphors developed for desktop based 3D virtual object manipulation. However these may not be appropriate for handheld phone based systems because of important differences between using a mobile phone 3D interface and a traditional desktop interface, including:

- Limited input options (no mouse/keyboard)
- Limited screen resolution
- Little graphics support
- Reduced processing power

There are also several key differences between using a mobile phone AR interface compared to a traditional head mounted display (HMD) based AR system, including:

- The display is handheld rather than headworn
- The phone affords a greater peripheral view
- The display and input device are connected

There are also some key differences between a mobile phone and a PDA. Mobile phones are operated using a one-handed button interface in contrast to the two-hand stylus interaction of the PDA. Due to the easy one-handed maneuvering it is possible to use the mobile phone as a tangible input object itself. In order to interact we can move the device relative to the world instead of moving the stylus relative a fairly static screen. Having one hand free allows the utilization of bimanual interaction techniques.

We assume that the phone is like a handheld AR lens giving a small view into the AR scene. We also assume that the user will be more likely move the phone-display than change their viewpoint relative to the phone. Thus the small form factor of the mobile phone lets us go beyond the looking-glass metaphor to an object-based approach. This metaphor can be applied to other AR applications that do not use a HMD, such as applications developed for projection screens,

tablet-PC and PDAs. Our input techniques are largely going to be based around motion of the phone itself, rather than keypad input into the phone.

4 Platform

In order to develop AR applications for Symbian based mobile phones there were several key steps we needed to perform:

- Port the ARToolKit tracking library to the Symbian operating system
- Develop a peer to peer communications layer
- Build a game application using 3D graphics
- Provide support for audio and haptic feedback

Henrysson was the first to implement ARToolKit for Symbian [2004]. To do this he wrote a C++ wrapper class in order to get rid of global variables, which are prohibited by Symbian. However, both the mobile phones we are targeting and the PDA used by Wagner lack a floating point unit, making floating-point arithmetic orders of magnitude slower than integer arithmetic. To overcome this, we wrote our own fixed-point library featuring variable precision. We did extensive performance tests to select the algorithms that ran fastest on the mobile phone. The average speed-up compared to corresponding floating-point functions was about 20 times. We started out by porting the functions rewritten by Wagner and continued backwards to cover most of functions needed for camera pose estimation. The resulting port runs several times faster than the original port. Some accuracy was lost when converting to fixed point but was perceived as acceptable.

Our graphics application was developed using OpenGL ES. In comparison to desktop OpenGL, memory and processor demanding functions such as 3D texturing and double precision floating point values have been removed along with GLU. A 16:16 fixed-point data type has been added to increase performance while retain some of the floating-point precision. The most noticeable difference is the removal of the immediate mode in favor of vertex arrays. Since Symbian does not permit any global variables the vertex and normal arrays must be declared constant, which limits the dynamic properties of objects.

The phone we were developing for, the Nokia 6630, ships with a software implementation of OpenGL ES. While this takes care of the low level rendering there is still need for a higher-level game engine with ability to import models created with 3D animation software and organize the content into a scene graph. Though M3G (JSR 184) provides model loading features it does not allow us to invoke the ARToolKit tracking library written in C++ since there is no equivalent to Java Native Interfaces (JNI) for J2ME. There are a few commercial game engines written in C++ but they are not suited for AR research applications that use calibration data and a tracking library to set the camera parameters.

To be able to import textured models from a 3D animation package we used a 3D converter application to exported the model to C++ code with OpenGL floating-point vertex arrays and then wrote a simple program that converted this into OpenGL ES compatible fixed point vertex arrays.

For our experiments in collaborative mobile phone AR we needed a way to transfer data between phones. We wrote a simple Bluetooth peer-to-peer communications layer. Our collaborative set-up consists of two mobile phones where one is a server that announces the game as a service and provides a channel for the client to connect to. The client makes an active search for the device and the service. There is thus no need for IP configuration.

Finally, we added support for audio and tactile feedback to our platform by using vibration and the media server from the Symbian API.

5 Featured Work 1: Object Manipulation and Scene Assembly

We need to develop input techniques that can be used one handed and only rely on a joypad and keypad input. Since the phone is handheld we can use the motion of the phone itself to interact with the virtual object. For example, as in AR-PAD, we can fix the virtual object relative to the phone and then position objects by moving the phone relative to the real world. Two handed interaction techniques can also be explored; one hand holding the phone and the second a the marker paper on which AR graphics are overlaid. This approach assumes that phone is like a handheld lens giving a small view into the AR scene. The small form factor of the phone lets us explore more object-based interaction techniques based around motion of the phone itself. Given these requirements there are several possible manipulation methods that could be tried. The following table shows the techniques we have implemented.

Positioning	Rotation
A Tangible 1: The object is fixed relative to the phone and moves when the user moves the phone. When released the object position is set to the final translated position while its orientation is reset to its original orientation.	A ArcBall: When the phone moves the relative motion of the phone is used as input into the arcball technique to rotate the currently selected object.
B Keypad/Joypad: The selected object is continuously translated in the X, Y or Z directions depending on the buttons currently held down.	B Keypad/Joypad: The object rotates about its own axis according to joypad and keypad input. Left and right joypad input causes rotation left and right about the vertical axis etc.
C Tangible 2: The same as Tangible 1, but the user can use bimanual input, moving both the phone and the object that the phone is tracked relative to.	C Tangible 1: The object is fixed relative to the phone and moves when the user moves the phone. When released the object orientation is set to the final phone orientation and position reset to its original position.
	D Tangible 2: The same as tangible 1, but the user can use bimanual input, moving both the phone and the object that the phone is being tracked relative to.

A user study with these techniques showed that the tangible translation was faster than the button interface, but most people felt that the keypad provided higher accuracy. For rotation the arcball and keypad interfaces were the fastest ones but there was no difference between the techniques when it came to perceived accuracy. For implementation details and the complete user study see the original paper [Henrysson et al. 2005a].

Based on these results we developed scene assembly application for the purpose of exploring how translation and rotation can be combined using both tangible and keypad interfaces. The application consists of a minimal scene with two boxes and a ground plane (see Figure 2). The boxes can be moved freely above the ground plane. In the center of the image plane are virtual cross hairs that are used for selection. Selection is made by pressing the joypad button when the box is in the cross hairs. The selection is based on a unique alpha value for each object and the selection is accomplished by sampling the alpha value of the central pixel, indicated by a crosshair. To indicate which object is selected, a yellow wireframe box is drawn around the object. When the joypad key is pressed the object is locked to the phone and highlighted in white. The virtual model is fixed in space relative to the phone and so can be rotated and translated at the same time. When the button is released the new transformation in the global (marker) space is calculated. The ambition for the keypad interface is for it to allow modification of all six degrees of freedom.

Figure 2: Ground plane and two boxes.

To switch between rotation and translation mode using the keypad interface, we have implemented a semi-transparent menu activated by pressing the standard menu button to the left of the joypad. By making the menu semi-transparent we allow the user to see the object to be transformed in the background. This will reduce the risk of forgetting which transformation to apply when browsing the menu. Since the selection is based on the alpha value of the central pixel, no selection can be made in menu mode and no object may have the same alpha value as the menu.

The menu layout consists of a 3 by 3 grid of icons that are mapped to the keypad buttons 1 to 9. (See Figure 3). The chosen transformation will by applied to the object highlighted by a yellow wireframe.

To translate the object in the x-z plane we use the four directions of the joypad and complement it with the 2 and 5 keys for translation along the y-axis. For rotation using the keypad we use the joypad to rotate around the x and z-axis, while the 2 and 5 buttons rotate the object around the y-axis.

5.1 Case study: Virtual Lego

So far we have only considered a minimal but general application allowing virtual block manipulation on a mobile phone. It can be used as a base for any 3D application where altering of the spatial relationship between objects are of interest. To demonstrate this we

Figure 3: The semi-transparent menu for selecting transformation mode

have implemented a simple virtual LEGO application (see Figure 4).

In this application the user can build structures by attaching virtual LEGO bricks to each other in any configuration that would be possible with the physical counterpart. The virtual bricks form substructures when attached to each other. These sub-structures can be treated as a group by selecting the bottom brick. The transformation made to this brick is propagated to the other brick in the sub-structure. This grouping into sub-structures is limited by the fact that a top brick cannot be attached to more than one bottom brick in the current implementation. However, one bottom brick can be the base for two or more top bricks. There is no restriction on how the number of bricks attached to each other.

When selected, the brick is detached from the brick below and can be moved freely. If other bricks are attached directly or indirectly to the selected brick, they will remain fixed in the local coordinate system of the selected brick.

Figure 4: Virtual LEGO bricks

Once released the application checks if the released piece is positioned within the margin of error to be attached to another piece. A grid restricts the transformations, making it easy to attach one piece on top of another as expected from the physical equivalent. We have not implemented any proper collision detection at this stage and the attachment is not checked continuously.

The phone vibrates when bricks are joined or pulled apart to give tactile feedback on detachment and attachment events.

The keypad interface works as before, but the transformation increments and decrements are adapted to the grid.

For further implementation details see the original paper [Henrysson et al. 2005c].

6 Featured Work 2: Collaborative AR

We have developed a simple tennis game to explore face-to-face collaborative AR on mobile phones. Tennis was chosen because it could be played in either a competitive or cooperative fashion, awareness of the other player is helpful, it requires only simple graphics and it is a game that most people are familiar with.

Our tennis application uses a set of three ARToolKit markers arranged in a line. These are printed on a piece of paper that is placed between the players. When the player points the camera phone at the markers they see a virtual tennis court model superimposed over the real world (see Figure 5). As long as one or more of these markers are in the field of view then the virtual tennis court will appear. This marker set is used to establish a global coordinate frame and both of the phones are tracked in this coordinate frame.

To serve the ball the player points their phone at the court and hits the 2 key on the keypad. Once the ball is in play there is no need to use the keypad any more. A simple physics engine is used to bounce the ball off the court and respond to when the player hits the ball with their camera phones

The simulation takes place in marker space. To check for possible collision with the racket, the position of the ball is transformed into camera space. This transformation is given by the ARToolKit tracking. The racket is defined as a circle centered on the z-axis in the xy-plane of the camera space. If there is an intersection between the racket plane and the ball, the direction of the z-axis is transformed into marker space and used to initialize the simulation.

By sending the direction and position vectors of the ball, the simulations will be synchronized each round. Both devices check for collision with the net and if the ball is bounced outside the court. If an incoming ball is missed the user gets to serve since the other devices Bluetooth is in listening mode. The simulation will always be restarted when data is sent and received. Each time the ball is hit there is a small sound played and the phone of the person that hits the ball vibrates, providing multi-sensory cues to help the players. We have not implemented score keeping yet, relying on players to keep score themselves. However this could be added in the future.

In order to evaluate the usability of mobile phones for collaborative AR we conducted a small pilot user study. We were particularly interested in two questions:

1 Does having an AR interface enhance the face to face gaming experience?

2 Is multi-sensory feedback useful for the game playing experience?

To explore these questions we conducted two experiments, both using the AR tennis game we have developed.

The user study showed that the AR was useful even though it was not necessary for the game to be playable. The users appreciated the multi-sensor feedback. However, sound turned out to be much more important than haptic feedback. For the complete user study see the original paper [Henrysson et al. 2005b].

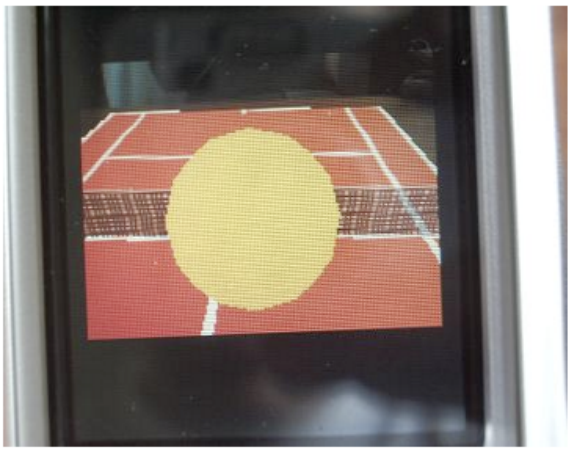


Figure 5: View of the tennis court.

7 Discussion

Even though AR is not essential for any of the presented applications we believe that using the video as background helps the users navigate in 3D. It also gives valuable feedback about the tracking. If the markers are lost the user can use the video feed to quickly maneuver the phone so that a marker becomes visible.

Tracking is the main limitation as the square must be visible at all times. We use multiple markers to extend the tracking range. This adds complexity to the calculations but we have managed to solve the associated problems. We have also experimented with motion flow tracking to allow one corner of the square marker to be outside the image, but this needs more work in order to be an enhancement. Possibly other sensors such as accelerometers or digital compasses could assist the tracking and together with a GPS module make outdoor mobile phone AR possible.

Our initial user experiences indicate that our manipulation set-up allows 6 DOF manipulation for scene assembly applications. By using an easily accessible menu we can map keys to axis instead of functions. Thus we can extend the interface to other operations such as scaling, cloning and various object specific features.

We believe our sample application can serve as a base for tabletop 3D applications where the spatial relationship between the objects is important. We assume most such applications will be games similar to the described virtual LEGO example, but some Virtual Reality applications that require 6 DOF could possibly be developed.

Our work in collaborative AR will be extended with such manipulation techniques rather than being limited to simple object intersections.

In developing a collaborative AR game for mobile phones we have learned a little about design guidelines that can be applied to future collaborative games:

- Face-to-face mobile games could benefit from adding AR interface technology.
- The use of multi-sensory feedback, especially audio is important for increasing game enjoyment.
- If visual tracking is used then the ideal games have a focus on a single shared game space, such as with our tennis game. This enables the players to easily see each other at the same time as the virtual content.

- Due to the slow tracking performance of the current generation of phones games should not rely on quick reflexes or fast competition.
- The screens on mobile phones are very small so collaborative AR games need only use a limited amount of graphics and should mainly focus on enhancing the face to face interaction.
- The use of an appropriate tangible object metaphor is also important for the usability of mobile phone AR applications. Physical manipulation of a phone is very natural so provides and intuitive interaction approach for collaborative AR games.

8 Conclusion and Future Work

In this paper we have presented some of our works in mobile phone AR that are the first of their kind. We developed a basic interaction application for 6DOF object manipulation and scene assembly on mobile phones using AR technology. We have also presented the first collaborative AR game for mobile phones and presented the results of our user studies, which might serve as design recommendations for others who want to develop 3D applications on mobile phones or PDAs. The user studies show that our platform is enough for creating an enjoyable multi-player game using only simple 3D graphics.

We will continue to explore the field of mobile phone AR and in the future we would like to employ the 6DOF manipulations in a collaborative set-up and conduct in-depth user studies. More applications will be developed to explore other aspects of mobile phone AR such as content creation and interfacing intelligent environments.

References

ARToolKit. www.hitl.washington.edu/artoolkit/.

CUTTING, D., ASSAD, M., CARMICHAEL, D. J., AND HUDSON, A. 2003. AR phone: Accessible Augmented Reality in the Intelligent Environment. In *OZCHI2003*.

FEINER, T. H. S., MACINTYRE, B., AND WEBSTER, T. 1997. A touring machine: Protoyping 3D mobile augmented reality systems for exploring the urban environment. In *ISWC (First IEEE International Symposium on Wearable Computers)*.

GEIGER, C., KLEINNJOHAN, B., REIMAN, C., AND STICHLING, D. 2001. Mobile AR4ALL. In *2nd IEEE and ACM International Symposium on Augmented Reality (ISAR 2001)*.

HACHET, M., POUDEROUX, J., AND GUITTON, P. 2005. A camera-based interface for interaction with mobile handheld computers. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, A. Press, Ed., 65–72.

HAKKARAINEN, M., AND WOODWARD, C. 2005. SymBall - Camera driven table tennis for mobile phones. In *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*.

HANSEN, T., ERIKSSON, E., AND LYKKE-OLESEN, A. 2005. Mixed Interaction Spaces Designing for Camera Based Interaction with Mobile Devices. In *Proceedings of CHI 2005*.

HENRYSSON, A., AND OLLILA, M. 2004. UMAR - Ubiquitous Mobile Augmented Reality. In *Proceedings of MUM 2004*, 41–46.

HENRYSSON, A., BILLINGHURST, M., AND OLLILA, M. 2005. Virtual Object Manipulation using a Mobile Phone. In *Proceedings of ICAT 2005 (to appear)*.

HENRYSSON, A., BILLINGHURST, M., AND OLLILA, M. 2005. Face to Face Collaborative AR on Mobile Phones. In *Proceedings of ISMAR 2005*, 80–89.

HENRYSSON, A., OLLILA, M., AND BILLINGHURST, M. 2005. Mobile Phone Based AR Scene Assembly . In *Proceedings of MUM 2005 (to Appear)*.

LARSEN, B., BÆRENTZEN, J., AND CHRISTENSEN, N. 2002. Using cellular phones to interact with virtual environments. In *ACM Siggraph 2002, Conference Abstracts and Applications*.

MOEHRING, M., LESSIG, C., AND BIMBER, O. 2004. Video See-Through AR on Consumer Cell Phones. In *International Symposium on Augmented and Mixed Reality (ISMAR'04)*, 252–253.

MOGILEV, D., KIYOKAWA, K., BILLINGHURST, M., AND PAIR, J. 2002. AR Pad: an interface for face-to-face AR collaboration. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, ACM Press, New York, NY, USA, 654–655.

REKIMOTO, J. 1996. Transvision: A Hand-held Augmented Reality System for Collaborative Deisgn. In *Virtual Systems and Multi-Media (VSMM)'96*.

ROHS, M. 2004. Real-World Interaction with Camera Phones. In *2nd International Symposium on Ubiquitous Computing Sysmtes (UCS2004)*.

WAGNER, D., AND SCHMALSTIEG, D. 2003. First steps towards handheld augmented reality. In *Seventh IEEE International Symposium on Wearable Computers*, IEEE, 127–135.

WAGNER, D., PINTARIC, T., LEDERMANN, F., AND SCHMALSTIEG, D. 2005. Towards Massively Multi-User Augmented Reality on Handheld Devices. In *Third International Conference on Pervasive Computing (Pervasive 2005)*.

Synthetic content approach for benchmarking mobile 3D graphics

Kari J. Kangas* Nokia
Mika Qvist† Nokia
Kari Pulli‡ Nokia

Abstract

This work-in-progress paper describes a synthetic content approach for measuring OpenGL ES 3D graphics performance of mobile devices. Our approach relies on a synthetic content tool that can create different kinds of OpenGL ES graphics content according to a large number of input parameters. The input parameters are obtained by analyzing real OpenGL ES content with an OpenGL ES tracer. The synthetic content is validated by comparing the performance of the real and synthetic contents in the same platform. Although we do not yet have all the required elements needed by our synthetic benchmark approach, initial studies have produced promising results which demonstrate that the synthetic content can match quite well the real content from the performance point of view.

1 Introduction

Being able to benchmark the OpenGL ES performance as early as possible in the design cycle of a mobile device is important. For example, the benchmark results can be used to guide performance optimization. Benchmarks can be used to understand the performance of a particular mobile device and how the performance changes with different content; this is especially important for understanding the performance of an OpenGL ES solution obtained from an outside supplier. We can also provide good estimates of the available graphics performance of an upcoming mobile device to content developers so that they can start designing content before they have seen the actual device.

However, benchmarking a device while it is still under development can be challenging. The most obvious benchmarking approach would be to use publicly available OpenGL ES games and other applications. Alternatively, we could use dedicated benchmark software such as FutureMark's SPMark04 or 3DMarkMobile06. These approaches, however, have their shortcomings. Third-party benchmark applications usually require completely integrated operating system with GUI support and such environments may not be available until quite late in the device development cycle. Software binary breaks may make it impossible to run existing binaries in new devices. The source code of benchmark applications may not be available, or if it is, it typically requires continuous, non-trivial porting efforts. Device manufacturers do not usually grant access to software development environments to outside companies until the product has been publicly launched, and some devices do not even support third party native applications. Finally, the OpenGL ES content in mobile devices ranges from simple 3D UI elements to Java M3G games to full-blown native OpenGL ES games. Therefore, using only few benchmark applications does not cover the expected range of OpenGL ES use cases very well.

In order to avoid these problems, we have adopted a synthetic benchmark content approach for measuring the OpenGL ES performance of mobile devices. Our approach relies on a special synthetic content tool that can create varying OpenGL ES graphics content according to a large number of input parameters. For measuring the OpenGL ES performance, the tool is executed on a target platform with selected input parameters. When executed, the tool draws the same synthetic content repeatedly and measures the steady-state frames-per-second (FPS) performance of the particular platform we are benchmarking.

The main input parameters for our synthetic content tool are triangle count, triangle size, and overdraw factor (how many times each pixel is written to, on the average). Triangles can be textured and we can control, for example, the texture type and texture filtering mode. The synthetic content tool has been written mostly using platform-independent ANSI C, with a special emphasis on making the software as easy to port as possible to different platforms. The OpenGL ES content used in benchmarks is created completely during run-time to keep the binary size small and to eliminate the need for artistic work.

The main problem with synthetic benchmark content is the question whether or not the synthetic content is similar enough (from the performance point of view) to the real content to be usable for benchmarking. The solution we present in this paper is that we measure features such as triangle count, average triangle size, and overdraw factor from real content, create synthetic benchmark content that has matching features, and then compare the performance of the real content and the synthetic content on the same platform. If the performance matches well enough, we conclude that we have successfully synthesized the real content.

Characterizing real-world 3D graphics workload for benchmarking purposes is described for example by Dunwoody and Linton [1990], Mitra and Chiueh [1999], and more recently by Antochi et al. [2004]. In our ongoing work, we are not focusing so much on analyzing the workload features, but more on the question how the analyzed workload features can be mapped into synthetic benchmark content so that the original workload and the synthetic workload are similar enough from the performance point of view. Our work resembles also the work done in render-time estimation (see for example [Funkhouser and Sequin 1993]) in a sense that we approximate real workload with a simplified synthetic workload (or model).

Although we do not yet have all the required elements needed by our synthetic benchmark approach, initial studies so far have produced promising results.

2 Creating synthetic benchmark content

The process of creating synthetic benchmark content is illustrated in Figure 1. We begin the synthetic benchmark content creation by running the OpenGL ES application we wish to synthesize on top of an OpenGL ES tracer. The tracer is a special version of OpenGL

*e-mail: kari.j.kangas@nokia.com
†e-mail: mika.qvist@nokia.com
‡e-mail: kari.pulli@nokia.com

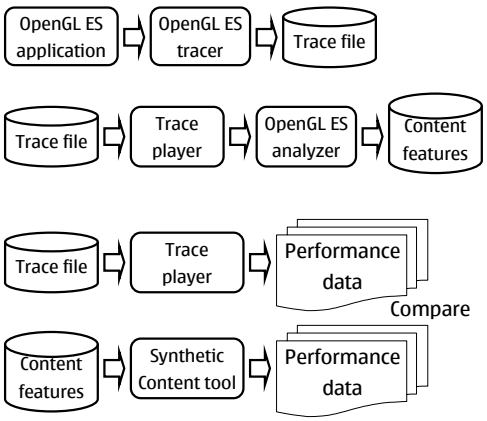


Figure 1: Synthetic benchmark content creation process.

ES library that captures the sequence of OpenGL ES calls, with the corresponding input parameters, into a trace file and then forwards the calls and parameters to the actual OpenGL ES library. To keep the size of the trace reasonable, we can instruct the tracer to capture only the OpenGL ES calls comprising the specific frames we are interested in subsequent analysis. This also allows us to minimize the delay caused by the trace capture as we can use a memory buffer of limited size for trace storage instead of constantly writing the trace into a memory card or similar mass storage.

The purpose of the tracer is to separate the OpenGL ES graphics calls from the OpenGL ES application. We can use the OpenGL ES trace player to replay the sequence of OpenGL ES calls, which recreates the OpenGL ES graphics in a controlled environment. For example, we can measure the performance of traced OpenGL ES content potentially on any OpenGL ES platform.

To get the content features such as triangle count, average triangle size, and overdraw factor for each frame in the traced OpenGL ES content, we feed the trace to an OpenGL ES analyzer. The OpenGL ES analyzer is a special version of the OpenGL ES library that keeps track of various content features used during rendering. For example, the analyzer records the number of incoming triangles, the average size of triangles (in fragments), and the total number of fragments written to the frame buffer for each OpenGL ES frame.

It is possible run the OpenGL ES application directly on top of the OpenGL analyzer to get the content features immediately as the application is run. However, the processing needed for the content analysis may interfere with the rendering so that some applications might, for example, skip frames to keep the rendering in sync with the audio playback. Alternatively, the analyzer can be implemented so that it analyzes the OpenGL ES calls directly without doing any actual rendering.

After the content features are extracted from the trace, we create synthetic benchmark content that has the matching content features. We can either make a matching synthetic frame for each original content frame, or we can make a single synthetic frame that represents a large combination of content frames. We can run the synthetic content tool on top of our OpenGL ES analyzer to make sure the content features are similar in both the original and synthetic content. Synthetic content tool will not skip any frames even if the rendering is very slow as the tool fully controls its own execution.

In order to verify and validate the synthetic benchmark content, both the synthetic the actual OpenGL ES trace are rendered on the same OpenGL ES platform. If the performance does not match within desired limits, we can refine the models in which the synthetic content tool creates OpenGL ES content from the input parameters. We can also modify the analyzer, if new types of input parameters are needed. Once we have successfully synthesized the content in one OpenGL ES platform, we can verify that the performance matches also on different platforms.

Eventually, the performance of the synthetic benchmark content should match the performance of the original OpenGL ES content on all tested platforms. When this happens, we can use it for benchmarking purposes instead of the original OpenGL ES content. We can even extrapolate from existing content so that we can test system responsiveness on content that has not been created yet.

3 Discussion and Future Work

We do not yet have a working OpenGL ES tracer, trace player, or analyzer. Instead, we have these tools for (desktop) OpenGL, and we have used them to analyze the content features of existing OpenGL games such as Quake and used the resulting features to create synthetic OpenGL ES benchmark content. The synthetic content tool works on top of both OpenGL (Win32) and OpenGL ES (Win32, Symbian, and PocketPC).

To provide some evidence that our current synthetic content tool can produce useful benchmark content, we have analyzed by hand the content features of one moderately complex OpenGL ES application (Nokia E3 2005 demo, see http://web.ngage.com/e3/video/videos.html?ID=22). We created matching synthetic benchmark content and compared the performance. The FPS performance of the real application was around 20 FPS whereas the FPS performance of the synthetic content was 24 FPS. This quick comparison should be considered only as an early indication that the performance of our synthetic benchmark content is roughly in the same ballpark as the real OpenGL ES with the matching content features.

During our work, we have found out that synthetic content tool is a very handy tool for experimenting how a particular OpenGL ES platform performs with different kinds of content. Our tool has a large set of easily configurable input parameters so creating a wide variety of synthetic content is easy. We have also developed tools for rapid presentation and comparison of the benchmark results. Both the easy content configurability and the ability to rapidly present the results have proved to be very valuable for understanding and especially communicating the 3D performance issues. As an example, if someone asks what happens to the performance once we change the texture filtering mode from nearest to bilinear and keep everything else as is, we can quickly measure it and present the results in an easily understandable format.

We have also investigated the possibility of using the trace as benchmark content. The main problem with trace is that the benchmark content (trace) cannot be modified very easily. For example we cannot easily modify the triangle count per frame and see how that affects the performance. However, once we analyze the content, we can change some parameters to create speculative benchmark content estimating possible future OpenGL ES applications, which we can use to see how the performance of a certain platform scales up.

Our future work comprises of developing the OpenGL ES tracer, trace player, and analyzer. After these tools are in place, we can properly analyze how well our synthetic benchmark content matches the source OpenGL ES content using more applications and different platforms. We are also studying different mechanisms of how the analyzed content features are used best to create syn-

thetic content with matching content features and similar performance characteristics.

We are also extending our synthetic content tool so that it can support other types of workloads besides OpenGL ES. For example, the synthetic content tool can generate synthetic CPU and memory load, play audio, and do game physics engine calculations while rendering OpenGL ES graphics. The synthetic content tool is designed so that diverse workloads can be mixed and matched easily within the same benchmark frame. Having this kind of mixture of workloads allows us to analyze the total system performance for complete and more realistic applications and use cases instead of concentrating on unrealistic graphics-only applications. As an example, we can easily test whether a particular phone model is capable of simultaneously transmitting and receiving wireless data at certain rate, playing mp3 with certain bit rate, calculating collision detection for three objects, and rendering graphics similar to Quake at 20 frame-per-second. By using a current analyzer, we can also easily measure how much power the phone consumes while doing all this. We feel that this flexibility allows us to use the synthetic content tool in scenarios that are not supported directly by other synthetic benchmark systems that concentrate only on one specific performance area such as 3D graphics performance.

We have also investigated the possibility of analyzing the high level structure of real-world OpenGL ES frames from the OpenGL ES trace. This structure could highlight for example how the application first draws the background and the background objects, followed by the foreground objects, followed by a special effects layer. We think that this structural information will allow us to better understand the important features of real OpenGL ES content. We also think that we could use the structural information to improve the quality of our synthetic content, for example by compositing the synthetic benchmark content frame from a set of synthetic content objects which are structured within a frame in a similar way as their counterparts in the OpenGL ES trace.

Finally, we are also extending our synthetic benchmark tool to support OpenVG benchmarking.

4 Conclusion

We have presented an approach and some preliminary results for estimating 3D performance of mobile devices while they are still under development and even when there is little existing 3D content available. Such a system is valuable both for engineers optimizing the design as well as for content creators preparing applications for the device. In both cases, performance estimates for applications could be made available even before hardware to run them exists. The crux of our future work lies in transferring our analysis tools from OpenGL to OpenGL ES, extending the analysis of content features and validating them with more experiments, and including other workloads such as the use of CPU for application logic and audio.

References

ANTOCHI, I., JUURLINK, B., VASSILIADIS, S., AND LIUHA, P. 2004. GraalBench: A 3D graphics benchmark suite for mobile phones. In *LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, ACM Press, New York, NY, USA, 1–9.

DUNWOODY, J. C., AND LINTON, M. A. 1990. Tracing interactive 3d graphics programs. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 155–163.

FUNKHOUSER, T. A., AND SEQUIN, C. H. 1993. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 247–254.

MITRA, T., AND CHIUEH, T. 1999. Dynamic 3d graphics workload characterization and the architectural implications. In *MICRO 32: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, IEEE Computer Society, Washington, DC, USA, 62–71.

The Orthogonal Constraints Problem with the Constraint Approach to Proxy-based Volume Haptics and a Solution

Karljohan Lundin* Matthew Cooper Anders Ynnerman

Norrköping Visualization and Interaction Studio
Linköping University, Sweden

Abstract

Recently the constraint approach to proxy-based volume haptics was introduced which provided a stable and effective means of conveying information about volumetric data through a haptic instrument. In this paper we present a proof that the approach is incapable of handling non-orthogonal constraints and discuss the implications of this restriction in detail. We also describe how full utilization of haptics applications in which multiple properties are used to enhance the understanding of complex data requires the use of non-orthogonal constraints. We then show how proxy-based volume haptics can be modified to allow for general constraints through the introduction of haptic primitives used to model the constraints. By balancing the forces exerted by the primitives on the proxy continuously, non-orthogonal constraints can be handled.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

Keywords: Proxy-based volume haptics, orthogonal constraints, haptic primitives

1 Introduction

It has been shown that adding haptic feedback to an application can significantly increase both precision and speed of human-computer interaction. Scientific visualization and data exploration are no exceptions. Most research on haptics focuses on surfaces, so the lack of surfaces in volume data demands alternative methods that do not require explicit surface representations.

The first method available for interaction with volumetric data was the force functions-based approach[Iwata and Noma 1993; Mor et al. 1996; Avila and Sobierajski 1996; Hashimoto and Iwata 1997; Infed et al. 1999; Lawrence et al. 2000]. With this method the force is expressed as a function of the data represented. It is easy to implement and therefore quite popular, but suffers from instability. Also, representing all features as simple forces of varying strength and direction can, in some cases, be considered to be a too simplistic approach.

As an alternative, the proxy-based approach to volume haptics was introduced in [Lundin et al. 2002]. It is more suitable for representing shapes in the volumetric data. The method generates simple haptic constraints, described in detail in section 2, that yield to a certain force. Using these constraints surface feedback can be generated from scalar density data by using the gradient information in the data. Since the constraints, and thus also the surfaces, yield to a certain force this method avoids introducing haptic occlusion; in other words it does not physically obstruct the exploration. The method has also been generalized[Ikits et al. 2003] to generate other shapes than surfaces and also include vector and tensor fields.

The proxy-based approach complements the force function-based methods by introducing "passive" interaction, as opposed to the continuous force from a simple function. While a force function calculates the feedback solely from the features found in the local data, a proxy-based method has the ability to generate feedback in response to user actions, so that if the user applies no force, no force is fed back.

In this paper we present a proof that the proxy-based approach breaks down if the constraints become non-orthogonal. It is thus incapable of handling non-orthogonal constraints without introducing severe haptic artifacts. We also describe the need for support of non-orthogonal constraints in haptic interaction of volumetric data and what limitations the lack of such support force on a haptic environment.

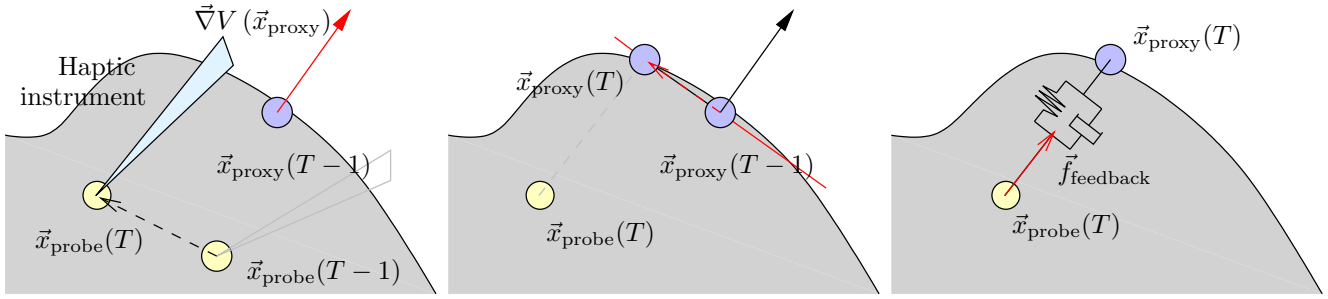
In [Lundin et al. 2005] we introduced haptic primitives as a way of modelling constraints from conceptually fundamental building blocks. By balancing the primitives it is also possible to include non-orthogonal constraints. In the second part of this paper we show how the use of haptic primitives, together with a numerical solver, enables the inclusion of general constraints in the haptic feedback loop, without the need for orthogonality.

2 Proxy-based Volume Haptics

In the proxy-based approach an internal proxy to the haptic probe is introduced. With the proxy, three simple steps are used in each time-frame of the haptic loop to produce the desired haptic effect from the constraints. First local data properties around the proxy point are determined. Then, depending on the local data, the proxy is moved a certain distance in local space. Finally the new proxy position is used to calculate the force feedback for the haptic instrument. These steps for proxy-based volume haptics are shown in figure 1.

1) Properties The continuous property fields needed to generate the haptic feedback are estimated from the discrete volumetric data through interpolation at the proxy position, see figure 1(a). For example the gradient vector field can be estimated from scalar data and curl or divergence may, if needed for generating the haptic effect, be extracted from

*e-mail: karlu@itn.liu.se



(a) Step 1: Evaluate local data properties. In this example the gradient vector, $\vec{\nabla}V(\vec{x}_{\text{proxy}})$, is extracted at the proxy position, $\vec{x}_{\text{proxy}}$.

(b) Step 2: Move the proxy point according to haptic constraints, in this case a to simulate the feeling of a plane.

(c) Step 3: Calculate feedback force by simulating spring-damper coupling.

Figure 1: Three steps for generating proxy-based haptic feedback, in this example from a virtual surface.

the vector data. These data are used in the second step to control the constraints that define the haptic effect. To produce more natural feedback from the volumetric data, transfer functions can be used to provide estimates of material properties directly from the data[Lundin et al. 2002; Avila and Sobierajski 1996; Aviles and Ranta 1999]. Even though using transfer functions for haptic feedback is not as established and extensively tested as using transfer functions for visual volume rendering, there is a close similarity between how they are used to define visual colours and to estimate material properties. Some examples of properties that are estimated and affect the haptic feedback are viscosity, friction, stiffness and flow strength.

2) Movements In the constraints approach simple one dimensional constraints, restricting the movement of the haptic instrument, are defined as functions of the properties of the local data. The strength of the constraint is controlled through a transfer function, as described above, and the direction of the constraint can be controlled by a vector property to produce a haptic representation of that property, for example the gradient. Each constraint controls the movement of the proxy in a direction to simulate a constraint in that direction. For a direction, represented by the unit vector $\hat{q}_i$, the equation that moves the proxy in the direction is formulated as

$$\vec{\eta} = \vec{x}_{\text{probe}} - \vec{x}_{\text{proxy}} \quad (1)$$

$$\vec{x}'_{\text{proxy}} = \vec{x}_{\text{proxy}} + \begin{cases} \hat{q}_i\,(\vec{\eta}\cdot\hat{q}_i - s_i/k)\,, & \text{if } s_i < k\,(\vec{\eta}\cdot\hat{q}_i) \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where $\vec{x}_{\text{proxy}}$ and $\vec{x}_{\text{probe}}$ are proxy and probe position, s_i is the strength of the constraint and k is the current stiffness used in the virtual coupling described below. By combining three independent orthogonal constraints in a local frame of reference a feeling of surfaces, friction, viscosity or transverse damping can be generated. The proxy movement is calculated separately in each direction and combined linearly to give the new proxy position, see figure 1(b).

3) Feedback After the new proxy position has been determined, the force feedback is calculated from a virtual spring-damper coupling the probe with the proxy point (see figure 1(c)). Thus the force feedback, $\vec{f}_{\text{feedback}}$, is evaluated through

$$\vec{f}_{\text{feedback}} = k\,(\vec{x}_{\text{proxy}} - \vec{x}_{\text{probe}}) + D\,(\vec{v}_{\text{proxy}} - \vec{v}_{\text{probe}}) \quad (3)$$

where $\vec{x}_{\text{proxy}}$ and $\vec{x}_{\text{probe}}$ are proxy and probe position, $\vec{v}_{\text{proxy}}$ and $\vec{v}_{\text{probe}}$ are proxy and probe velocity and k and D are stiffness and damper parameters, respectively.

3 Non-orthogonality Problem

As has already been mentioned, the above outlined constraint approach to proxy-based volume haptics is incapable of handling non-orthogonal constraints correctly. In this section we present a proof of this statement and discuss the impact and consequences of this.

3.1 Current Limitations

The constraint approach to proxy-based volume haptics will produce severe haptic artifacts when non-orthogonal features are encountered. This can be easily proven using a simple example with three constraints of zero strength. With no strength on the constraints (and a non zero stiffness, k) the proxy should end up at the probe position, so that the feedback from equation 3 yield zero.

Orthogonality requirement. Here it is shown that this requires orthogonality between the different constraints. From equation 2, by setting the strength, s_i, to zero and applying it three times, in different directions, we get

$$\vec{\eta} = \vec{x}_{\text{probe}} - \vec{x}_{\text{proxy}} \quad (4)$$

$$\vec{x}'_{\text{proxy}} = \vec{x}_{\text{proxy}} + \hat{q}_1\,(\vec{\eta}\cdot\hat{q}_1) + \hat{q}_2\,(\vec{\eta}\cdot\hat{q}_2) + \hat{q}_3\,(\vec{\eta}\cdot\hat{q}_3) \quad (5)$$

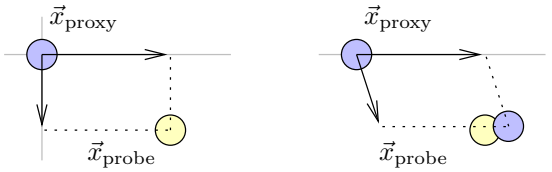
For the proxy, $\vec{x}_{\text{proxy}}$, to end up at the probe position, $\vec{x}_{\text{probe}}$, we see that

$$\begin{aligned} \vec{x}'_{\text{proxy}} &= \vec{x}_{\text{probe}} \\ &= \vec{x}_{\text{proxy}} + \vec{x}_{\text{probe}} - \vec{x}_{\text{proxy}} \\ &= \vec{x}_{\text{proxy}} + \vec{\eta} \end{aligned} \quad (6)$$

By combining equations 5 and 6 we get

$$\vec{\eta} = \hat{q}_1\,(\vec{\eta}\cdot\hat{q}_1) + \hat{q}_2\,(\vec{\eta}\cdot\hat{q}_2) + \hat{q}_3\,(\vec{\eta}\cdot\hat{q}_3) \quad (7)$$

which can only be true if the unit vectors, $\hat{q}$, are orthogonal. Thus, to get zero feedback from constraints with zero



(a) Orthogonal constraints move the proxy to the probe position.

(b) Non-orthogonal constraints move the proxy to an incorrect position.

Figure 2: Proxy movements with two constraints using the constraint-based approach. Both constraints have zero strength in this example and the proxy should be moved to the probe position.

strength, each constraint has to be orthogonal to the other constraints. □

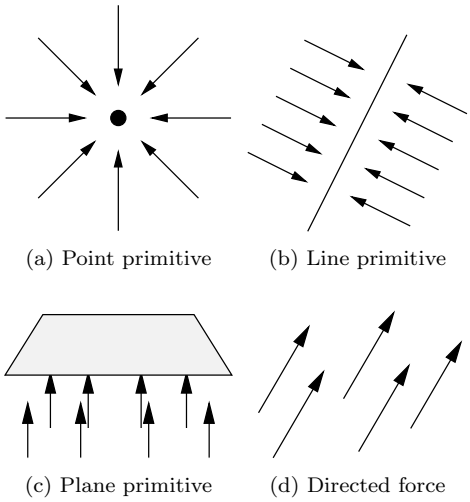
The individual movements due to non-orthogonal constraints will contribute to each other, which results in a different movement than desired. This effect is shown in figure 2. All combinations of non-orthogonal constraints show this behaviour and there is no simple way to circumvent this problem.

3.2 Impact

In research on and applications of haptics in scientific visualization, single properties have often been used to generate feedback from a single dataset at a time. Interacting with a single object at a time is common in visualization and exploration applications, however, as we move closer to immersive virtual reality environments this might change. In virtual reality (VR) applications it is common that the user interacts with multiple objects at a time. This has not been identified as a problem, since multiple surface objects can be easily handled by a user and it is the inter-object collision handling that has been the most challenging problem. With the constraint approach to volume haptics, however, interaction with volumetric data is restricted to single objects at a time. Thus as volume visualization and haptic force feedback are combined in virtual environments, the constraint approach will not suffice.

The biggest problem is, however, in the interaction with single objects of more advanced nature. Scientific visualization of multi-modal data is a growing area in which multiple datasets of the same object but of different modalities are co-located and co-registered to provide more information about the object than is possible with a single modality. An example is Computational Fluid Dynamics that produces both vector data describing the flow and several extra scalar datasets describing pressure, density, temperature, etc. The features of the different modalities are not guaranteed to be mutually orthogonal. Thus, the requirement of orthogonal constraints in the haptic interaction makes feedback from multi-modal data impossible.

Also in interaction and exploration of a single dataset there may be multiple properties that can be used to produce simultaneous information feedback. One example of this is the virtual wind-tunnel, where several properties of the vector data alone are interesting, such as the path of the flow, the strength of the flow and the vorticity. Since the constraint approach is incapable of handling non-orthogonal constraints simultaneously, the user must select only a single property to provide both information about the data and guide the user through the exploration process.



(a) Point primitive

(b) Line primitive

(c) Plane primitive

(d) Directed force

Figure 3: Forces from the haptic primitives.

4 Haptic Primitives

So far we have proved the existence of the orthogonality-problem and discussed its impact. In this section we describe our solution to the problem.

Using the single-dimensional constraints, more advanced constraints and types of behaviour are generated by rotating the constraints with respect to the haptic instrument. Our first step in the process of removing the orthogonality-problem is, instead, to provide one type of constraint for each desired type of haptic behaviour. The most basic way to discriminate types of constraints is by their dimensionality — constraints of one, two and three degrees of freedom. We call these *plane*, *line* and *point*, respectively, from the shape of their respective domain of constraint free motion. By adding a fourth type of haptic effect, *force*, we have a set of basic components that can be used to build any of the previously encountered haptic effects. We call these base components *haptic primitives*[Lundin et al. 2005].

The primitives provide a high level of abstraction that can be used to model a wide range of haptic modes and combinations of such, representing data from different scientific disciplines. They are designed to pull and push the haptic probe in well-defined directions to simulate features in the volume, see figure 3. As we will show this is done by controlling the proxy point movements. The haptic primitives and their properties are thus used to generate haptic feedback in a manner similar to that of how the constraints are used.

The haptic primitives can be expressed as simple force functions as shown below. It should be noted that the forces originating from the primitives act only on the proxy and are a means to find the new proxy position. This is done by balancing the net force from the primitives against the force feedback from equation 3 (see example in figure 4). This is presented in detail in section 4.2. After the new proxy position is found, the force feedback is calculated through the virtual coupling, equation 3.

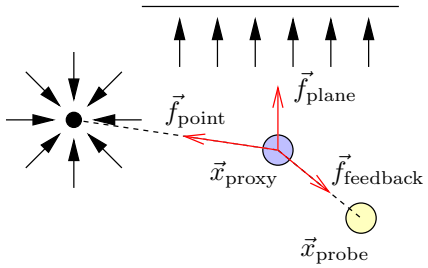


Figure 4: Finding the proxy position that balances the force between plane and point primitive ($\vec{f}_{\text{plane}}$ and $\vec{f}_{\text{point}}$, respectively) and feedback force ($\vec{f}_{\text{feedback}}$).

4.1 Intermediate Force Representation

In this section we describe the haptic primitives and how they, from their position and strength, define the force field that affects the proxy. For primitive i at position $\vec{x}_i$ having a strength of s_i, if the primitive has a direction/orientation then this is defined by a unit vector, $\hat{q}_i$. The proxy position is, as before, denoted $\vec{x}_{\text{proxy}}$.

The directed force is the simplest primitive. It has no position and generates a force $\vec{f}_i$ defined by

$$\vec{f}_i\left(\vec{x}_{\text{proxy}}\right) = s_i \hat{q}_i \tag{8}$$

This primitive can be used to simulate gravity or magnetic attraction. It also provides a means for integrating force-function feedback with the proxy-based approach.

The point primitive attracts the proxy towards the position of the primitive. With the displacement of the proxy relative to the primitive being $\vec{x}_i - \vec{x}_{\text{proxy}}$, we calculate the force by

$$\vec{f}_i\left(\vec{x}_{\text{proxy}}\right) = \begin{cases} \vec{0}, & \text{if } |\vec{x}_i - \vec{x}_{\text{proxy}}| = 0 \\ s_i \frac{\vec{x}_i - \vec{x}_{\text{proxy}}}{|\vec{x}_i - \vec{x}_{\text{proxy}}|}, & \text{if } |\vec{x}_i - \vec{x}_{\text{proxy}}| \neq 0 \end{cases} \tag{9}$$

The uniform feedback from this primitive makes it suitable for such effects as viscosity.

The line primitive has both position, strength and orientation. It attracts the proxy towards the closest point on the line defined by the position and the direction vector $\hat{q}_i$. With a vector $\vec{m}$, pointing from the proxy to the closest point on the line, being expressed by

$$\vec{m} = \hat{q}_i \left[\hat{q}_i \cdot \left(\vec{x}_{\text{proxy}} - \vec{x}_i\right)\right] - \left(\vec{x}_{\text{proxy}} - \vec{x}_i\right) \tag{10}$$

we calculate the force by

$$\vec{f}_i\left(\vec{x}_{\text{proxy}}\right) = \begin{cases} \vec{0}, & \text{if } |\vec{m}| = 0 \\ s_i \frac{\vec{m}}{|\vec{m}|}, & \text{if } |\vec{m}| \neq 0 \end{cases} \tag{11}$$

The plane primitive is most similar to the simple constraint in that it produces a yielding restraining force in one single direction. To generate this effect it attracts the proxy in the direction of the surface normal, $\hat{q}_i$, but only when the proxy is on the negative side of the surface, i.e. when

$$\left(\vec{x}_{\text{proxy}} - \vec{x}_i\right) \cdot \hat{q}_i < 0 \tag{12}$$

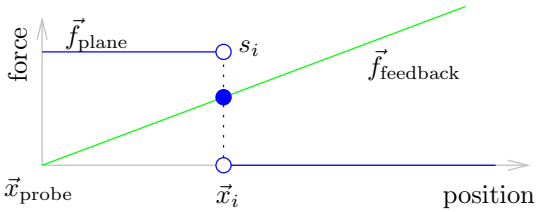


Figure 5: Finding the static equilibrium with a plane primitive — balance between the force feedback from equation 3 and the plane primitive at position x_i, equation 13.

As long as equation 12 holds, the force from the plane primitive is constant, so we define the force by

$$\vec{f}_i\left(\vec{x}_{\text{proxy}}\right) = \begin{cases} 0, & \text{if } \left(\vec{x}_{\text{proxy}} - \vec{x}_i\right) \cdot \hat{q}_i \geq 0 \\ s_i \hat{q}_i, & \text{if } \left(\vec{x}_{\text{proxy}} - \vec{x}_i\right) \cdot \hat{q}_i < 0 \end{cases} \tag{13}$$

4.2 Finding the new Proxy Position

To simulate the haptic feedback, the proxy is moved to a new position in every time-frame. This new position is found by balancing the force feedback from equation 3 with the force from the haptic primitives involved, as is shown in figure 4. In our work we disregard the damping term to simplify the calculations, that is we set $D = 0$ in equation 3. Suppose that we have sets of directed force, point, line and plane primitives denoted $\mathcal{A}_{\text{directed}}$, $\mathcal{A}_{\text{point}}$, $\mathcal{A}_{\text{line}}$ and $\mathcal{A}_{\text{plane}}$, respectively. We calculate the residual force, $\vec{f}_{\text{res}}$, from the primitives and the force feedback by

$$\begin{aligned} \vec{f}_{\text{res}} &= -k\left(\vec{x}_{\text{proxy}} - \vec{x}_{\text{probe}}\right) \\ &+ \sum_{i \in \mathcal{A}_{\text{directed}}} s_i\, \hat{q}_i \\ &+ \sum_{i \in \mathcal{A}_{\text{point}}} \begin{cases} 0, & \text{if } |\vec{x}_i - \vec{x}_{\text{proxy}}| = 0 \\ s_i \frac{\vec{x}_i - \vec{x}_{\text{proxy}}}{|\vec{x}_i - \vec{x}_{\text{proxy}}|}, & \text{if } |\vec{x}_i - \vec{x}_{\text{proxy}}| \neq 0 \end{cases} \\ &+ \sum_{i \in \mathcal{A}_{\text{line}}} \begin{cases} 0, & \text{if } |\vec{m}| = 0 \\ s_i \frac{\vec{m}}{|\vec{m}|}, & \text{if } |\vec{m}| \neq 0 \end{cases} \\ &+ \sum_{i \in \mathcal{A}_{\text{plane}}} \begin{cases} 0, & \text{if } \left(\vec{x}_{\text{proxy}} - \vec{x}_i\right) \cdot \hat{q}_i \geq 0 \\ s_i \hat{q}_i, & \text{if } \left(\vec{x}_{\text{proxy}} - \vec{x}_i\right) \cdot \hat{q}_i < 0 \end{cases} \end{aligned} \tag{14}$$

The new proxy position is then found by solving

$$\vec{f}_{\text{res}}\left(\vec{x}_{\text{proxy}}\right) = \vec{0} \tag{15}$$

Equation 15 is solved using a numerical solver that searches for a best match by minimizing the magnitude of the function. The minimum magnitude of the residual force will yield the same proxy position as an analytical solution would (see figure 5).

As an example of the effects produced by our approach consider only the plane primitive in balance with the force feedback: the proxy will be positioned on the plane as long as the force feedback projected on the plane normal is less than or equal to the strength of the plane primitive. Then the setup behaves just like surface haptics, see figure 6, steps a to c. When the probe is moved further away from the plane, the proxy will find an equilibrium below the plane, giving the same effect as the constraints of earlier methods, see figure 6, step d. A similar effect is also generated by the point and the line primitive.

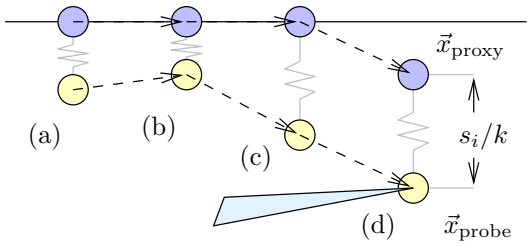


Figure 6: How the proxy is moved over a plane primitive when the probe is moved. The maximum distance between the proxy and probe positions induced by the haptic primitive is given by the primitive strength divided by the stiffness in equation 3, i.e. s_i/k.

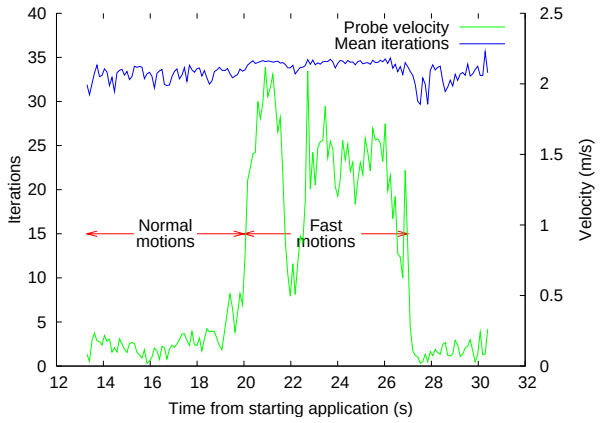


Figure 7: Number of iterations needed for our numerical solver to converge during a haptic exploration. The velocity of the probe is also presented.

4.3 The Numerical Solver

The solver we use to find the best match is a simple Euler solver. The problem is a non-linear minimization problem that can be solved using alternative iterative methods, such as the Nelder-Mead Solver. We have, however, so far found no analytical solution.

The precision and performance of the numerical solver is of utmost importance since it is the part of the algorithm that determines the proxy position for each time-frame and is the most computationally intensive part of the system. To speed up the convergence for the solver while maintaining high precision for the result we deploy an adaptive step-length approach. For two haptic primitives we find the solution in, typically, 30–40 iterations which takes less than 100 μs on our current hardware. The time complexity of the solver with respect to the number of haptic primitives is $\mathcal{O}(N)$. A graph describing the solver behaviour over a haptic exploration can be found in figure 7.

5 Conclusions

We have shown that the current constraint-oriented proxy-based approach to volume haptics has a severe limitation: all features represented by the haptic feedback *must* be orthogonal for the algorithm to work. A wide adoption of haptics for scientific visualization needs a solution to this problem. With this limitation haptic feedback can not be generated from an interaction with multiple objects or with multi-modal data.

To solve the problem, all constraints must be handled concurrently. This is done in our new primitive-based algorithm for volume haptics, introduced in [Lundin et al. 2005]. The method uses the notion of haptic primitives both as a comprehensive abstraction layer for implementing haptic schemes and as an effective means of calculating the haptic force feedback. Our solution to the orthogonal constraints problem allows for simultaneous feedback from multiple objects, haptic interaction with multi-modal data and simultaneous haptic interaction with multiple properties from a single dataset.

References

Avila, R. S., and Sobierajski, L. M. 1996. A haptic interaction method for volume visualization. In *Proceedings at IEEE Visualization*, 197–204.

Aviles, W., and Ranta, J. 1999. Haptic interaction with geoscientific data. In *Proceedings at Phantom User Group Workshop'99*.

Hashimoto, W., and Iwata, H. 1997. A versatile software platform for visual/haptic environment. *Journal of Control, Automation and Systems Engineering*, 106–114.

Ikits, M., Brederson, J. D., Hansen, C. D., and Johnson, C. R. 2003. A constraint-based technique for haptic volume exploration. In *Proceedings of IEEE Visualization '03, pp. 263–269*.

Infed, F., Brown, S. V., Lee, C. D., Lawrence, D. A., Dougherty, A. M., and Pao, L. Y. 1999. Combined visual/haptic rendering modes for scientific visualization. In *Proceedings of 8th Annual Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Iwata, H., and Noma, H. 1993. Volume haptization. In *Proceedings of IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, 16–23.

Lawrence, D. A., Lee, C. D., Pao, L. Y., and Novoselov, R. Y. 2000. Shock and vortex visualization using a combined visual/haptic interface. In *Proceedings of IEEE Conference on Visualization and Computer Graphics*.

Lundin, K., Ynnerman, A., and Gudmundsson, B. 2002. Proxy-based haptic feedback from volumetric density data. In *Proceedings of Eurohaptics*, University of Edinburgh, United Kingdom, 104–109.

Lundin, K., Gudmundsson, B., and Ynnerman, A. 2005. General proxy-based haptics for volume visualization. In *Proceedings of the World Haptics Conference*, IEEE, 557–560.

Mor, A., Gibson, S., and Samosky, J. 1996. Interacting with 3-dimensional medical data: Haptic feedback for surgical simulation. In *Proceedings of Phantom User Group Workshop'96*.

A Novel Approach to Compress Reflection Functions Based on PCA

Björn Olsson*
Linköping University

Anders Hast†
University of Gävle

Anders Ynnerman‡
Linköping University

Figure 1: Three examples of generated images using 25 coefficients.

Abstract

In many applications it is important to perform image-based relighting. That is, to synthesize a scene in various lighting conditions without explicitely rerendering the image. This paper proposes an efficient compression method, which after a precomputing step allows an efficient re-rendering of the scene. The best results are achieved on scenes with a limited number of materials, but it may also be used on arbitrary scenes. In this work images of a static scene are generated and the method is exemplified using a dataset of ray-traced images.

CR Categories: I.3.3 [Computer graphics]: Picture/Image Generation; I.3.7 [Computer graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing and texture

Keywords: BRDF, PCA, image-based relighting

1 Introduction

Image-based relighting is important in many applications, for example in the entertainment industry and in flight-simulators. This technique is based on methods developed decades ago and the first renderings with reflection mapping were performed by [Blinn and Newell 1976]. In the years to follow the ideas were developed further by many people and were formalized by [Greene 1986].

A very large amount of work has been performed in the field of relighting. In [Kristensen et al. 2005] a general method for real-time relighting of scenes was presented. This method used the concepts of unstructered light clouds and clustered PCA to render scenes with moving lights and dynamic cameras. Another example is the method presented by [Wood et al. 2000], who used a surface light-field to generate images of shiny objects at arbitrary lighting conditions. Another method was presented by [Nimeroff et al. 1994], who used basis images for efficient re-rendering of a scene.

The PCA method has previously been used to render transparent objects (See [Matusik et al. 2002]). In contrast to earlier methods the approach presented in their paper computed the principal components for global reflection maps. In [Matusik et al. 2002] an approach similar to JPEG-compression was used. The reflection maps were divided into zones of 16x16 pixels and each zone was compressed to a small number of coefficients. Another paper by [Epstein et al. 1995] investigated the number of eigenimages required to generate the scene. In this case global eigencomponents computed from a data set of variably lit images were used. A related approach, the eigentexture method, was described by [Nishino et al. 1999].

Another approach, matrix radiance transfer, which also builds on PCA compression of BRDFs was presented by [Lehtinen and Kautz 2003]. In [Ho et al. 2003] a related approach was described. The PCA method was applied on a set of reference images with the same view, but with different illumination conditions. However, in contrast to the method presented in this paper PCA was applied block-wise. In [Shim and Chen 2005] a statistical approach, which can be used to compare various methods estimating surface reflection functions was described. A method related to the approach presented in this paper was described in [Sloan et al. 2002]. In that paper the reflection functions were represented using spherical harmonics. CPCA (Clustered principal component analysis) is another way to compress precomputed radiance transfer, which was used in [Sloan et al. 2003]. The algorithm was implemented on graphics hardware. An interesting method using CPCA can be found in [Liu et al. 2004]. It was used to perform relighting on large models.

In this paper a new reconstruction technique applied on reflection functions is described. The goal of this work has been to develop a simple method, which is fast and efficient. This approach builds on the PCA method and is especially efficient for scenes with a limited number of textures. For the method presented in this paper the viewpoint is fixed relative to the object and a set of photographs is captured while the light source is repositioned for each photograph. In addition this representation allows a very efficient reconstruction step due to the linear basis in the PCA method. After a precomputation step the scene can be relighted in arbitrary lighting conditions.

By using this method a reflection map can be characterized with a small number of coefficients. In this paper the method is applied to still images, but it would be very easy to generalize the method to arbitrary views by predicting the image appearance for several directions and interpolating the current view. The images used in this work are ray-traced, but it would also be possible to use captured

*email:Bjorn_A_Olsson@hotmail.com
†email:Anders.Hast@hig.se
‡email: andyn@itn.liu.se

images. In the next section the method is described in detail. The results and a discussion follows.

2 Method

The method proposed in this paper was applied to one data set of ray-traced images. After the precomputation and configuration phase it was possible to generate images of arbitrary lighting conditions without explicitely resynthesizing the scene. The PCA-method was used to compress the reflection functions to small coefficient vectors.

2.1 Principal Component Analysis

The principal component analysis method (PCA for short), also named the Hotelling transform (See for example Joliffe [Joliffe 2002] or Haykin [Haykin 1999]) is a general method to compress input data to smaller representations by computing the major components of the input-data. This technique is common in statistics as well as in image processing.

The PCA method is used to decompose images into a number of basis components. The pixels in an image are rearranged into a vector $\bar{x} = [x_1, x_2, \ldots, x_N]$ of length N. We assume that we have M such vectors. The transpose is denoted $\vec{x}^T$. The mean vector $\overline{m_x}$ is defined as $\overline{m_x} = E(\bar{x})$ and the covariance matrix is $\mathbf{C_{xx}} = E((\bar{x} - \overline{m_x}) \cdot (\bar{x} - \overline{m_x})^T)$, where $E(.)$ means the expectation value of a stochastic variable. The eigenvalue problem is then defined $\mathbf{C_{xx}V} = \mathbf{VD}$, where $\mathbf{V}$ is a matrix with the eigenvectors as columns and $\mathbf{D}$ a diagonal matrix with the eigenvalues along the main diagonal.

For the resolution needed the size of the covariance matrix becomes impractical. The size of the matrix can be decreased by using dimensionality reduction [Haykin 1999]. Let $\mathbf{Y}$ be the rectangular data matrix of size MxN. The matrix has the same height as the number of vectors (M). The eigenvalue problem, $\mathbf{C_{xx}V} = \mathbf{VD}$, can be written as

$$\mathbf{C_{xx}} = \frac{1}{N^2}\mathbf{Y^T Y} \Rightarrow \frac{1}{N^2}\mathbf{Y^T YV} = \mathbf{VD} \tag{1}$$

The size of the covariance matrix is reduced from a size of NxN elements to a size of MxM elements by multiplying with $\mathbf{Y}$ from the left and changing variable $\mathbf{W} = \mathbf{YV}$.

$$\frac{1}{N^2}\mathbf{YY^T YV} = \mathbf{YVD} \Rightarrow \frac{1}{N^2}\mathbf{YY^T W} = \mathbf{WD}. \tag{2}$$

We have now computed the eigenvectors in $\mathbf{W}$, which is a sub-space of $\mathbf{V}$. To transform them to the original system $\mathbf{W}$ is multiplied with $\mathbf{Y^T}$.

$$\mathbf{W} = \mathbf{YV} \Rightarrow \mathbf{Y^T W} = \mathbf{Y^T YV} = \mathbf{C_{xx}V} \Rightarrow \mathbf{Y^T W} = \mathbf{VD} \tag{3}$$

$\mathbf{VD}$ are the eigenvectors multiplied with the eigenvalues. This scaling-factor does not alter the result since the principal components are normalized before usage

$$\overline{\phi(k)} = \frac{\mathbf{V}(:,k)}{\sqrt{\mathbf{V}(:,k) \cdot \mathbf{V}(:,k)^T}}. \tag{4}$$

An image is transformed to a coefficient vector by

$$\overline{\kappa(k)} = (\bar{x} \cdot \overline{\phi(k)}^T), k = 1, \ldots, N \tag{5}$$

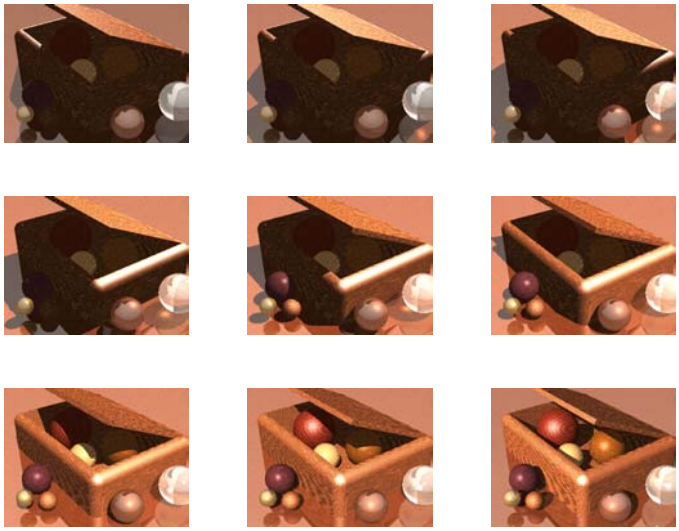


Figure 2: A number of original images for varying lighting conditons are exemplified in this figure.

This transformation is denoted $\bar{x} \overset{PCA}{\rightarrow} \overline{\kappa}$. The inverse transform $\overline{\kappa} \overset{PCA^{-1}}{\rightarrow} \bar{x}$ is defined by

$$\bar{x} = \sum_{k=1}^{N} \overline{\kappa(k)} \cdot \overline{\phi(k)} \tag{6}$$

In practical applications vector $\overline{\kappa}$ is often truncated and only the P most important coefficients are computed

$$\overline{\kappa_c(k)}, k = 1, \ldots, P \tag{7}$$

In this case the inverse transform will be inexact

$$\overline{\kappa_c} \overset{PCA^{-1}}{\rightarrow} \tilde{x}. \tag{8}$$

2.2 Precomputing and synthesizing

The method can be divided into the precomputing and synthesizing phases:

Precomputing Reflection maps, $\mathbf{R_{xy}}$, one for each pixel, are computed from the input images. These reflection maps have size QxQx3 elements. The PCA method is used to compress the reflection maps to small coefficient vectors, one vector for each pixel.

Synthesizing The scene is relighted for a specific lighting condition. This is performed by relighting each individual pixel by using the corresponding reflection function and the lightmap. By using the PCA approach it is possible to perform the computation much more efficiently.

2.3 Precomputing

The initial configuration phase can be divided into a number of steps:

Collect data set This method uses a database of images containing images of one identical scene lighted by a point source

with varying direction. The database is used to construct a method to relight the scene with arbitrary light sources. The data set can contain either captured or ray-traced images, but the best results are achieved for scenes with a limited number of surface textures. Since this work has concerned synthetic images, the colour channels are assumed to behave linearly.

Discretize the lighting directions Only the upper semi-sphere is considered in this work. It is discretized in a limited number of directions. A circular bitmap image with varying diameter is used as a model of the reflection map. Each pixel is associated with a corresponding direction. By using this technique interpolation artifacts are avoided. The directions are stored separately to be used in the generating process.

Compute reflection maps Each generated image is associated with a specific direction. For each pixel a separate reflection map is computed. This is performed by picking one pixel from each generated image and locating them in the associated positions. The procedure is exemplified in Figure 3, in which the pixel-information of the images is transformed to the reflection functions. The reflection matrices are stored in a 5D data structure (image_x, image_y, r_map_x, r_map_y, {rgb}).

Rotation of reflection maps All reflection maps are rotated along their major axes. The colour planes are not rotated individually, but instead the intensity of each pixel is computed by $I = R + G + B$. The major axis of the corresponding binary image is computed by thresholding the intensity levels. Pixels with $I(x,y) > threshold$ are included in the calculation.

The major axis θ is computed by the following method:

- $f_{xx} = 0$, $f_{yy} = 0$ and $f_{xy} = 0$.
  1. $\Delta x = x - m_x$, where x is the current x coordinate and m_x the mean x position.
  2. $\Delta y = y - m_y$, m_y is the mean y position and y the current y coordinate.
  3. $f_{xx} = f_{xx} + \Delta x \cdot \Delta x$
  4. $f_{yy} = f_{yy} + \Delta y \cdot \Delta y$
  5. $f_{xy} = f_{xy} + \Delta x \cdot \Delta y$

  Then perform the following calculations:
- $p = -(f_{yy} + f_{xx})$
- $q = f_{xx} \cdot f_{yy} - f_{xy} \cdot f_{xy}$
- $f = -\frac{p}{2} + \sqrt{\frac{p \cdot p}{4} - q}$
- The mean axis will be $\theta = \frac{q}{f}$

All reflection maps are rotated to have their major axes in the same direction. The rotation angles are stored in a separate matrix, which is used in the synthesizing phase.

Compute the eigencomponents In the next step the PCA method is used to compute a number of the most important eigencomponents, $\phi_{\mathbf{1}}, \ldots, \phi_{\mathbf{P}}$ from the reflection maps. A large number of reflection maps is needed to compute sharp components.

Compute coefficients The reflection maps are projected one by one onto the most important eigencomponents and the coefficients are stored. The result will be a coefficient matrix with one coefficient vector for each pixel of the image, $\kappa(\mathbf{1}, \ldots, \mathbf{P}; \mathbf{1}, \ldots, \mathbf{M})$, where P is the number of coefficients and M is the number of pixels.

Transform the lightmap to this specific representation The original lightmap defining the lighting conditions is rescaled to L_{xy} of size QxQ, where Q is the size of the reflection map.

Precompute the rotated eigencomponents: $\phi(1, \ldots, N, 1 : \sigma : 180)$ To make the computations more efficient, rotated versions of the eigencomponents are computed before the calculations. The angle is discretized in steps of size σ. In the calculations the eigencomponent closest to the angle is used. If the reflection functions are of a small size the error introduced will be limited.

Image representation The result will be a coefficient matrix with P coefficients for each pixel and a data structure with the rotated eigencomponents. In the beginning of the synthesizing phase the eigen-numbers are computed by using the appropriate lightmap.

2.4 Synthesizing

In the synthesizing phase the data volume in Figure 5 is used. One column consisting of a coefficient vector and a rotation vector is applied to construct one reflection function.

Collect lighting conditions Acquire an HDR-fisheye representation of the lighting conditions.

Rescaling Recompute the light representation to this specific discretization. The result will be a QxQx3 matrix, $\mathbf{L_{xy}}$. In the synthesizing phase the data volume in Figure 4 is used. One column consisting of a coefficient vector and a rotation vector is applied to construct one reflection function.

Algorithm In this section the principle behind the algorithm is presented. It is described in more detail in the upper row of Figure 4. One pixel of the resulting image is generated at a time:

1. Generate a reflection function $\mathbf{R_{xy}}$ from the corresponding coefficient vector $\overline{\kappa}$ for pixel ψ.

$$\mathbf{R_{xy}} = \sum_{k=1}^{P} \overline{\kappa(k)} \cdot \phi(\mathbf{k}), \tag{9}$$

   P is the number of coefficients and $\phi(\mathbf{k})$ the eigenvectors.

2. Rotate the reflection map angle θ.

$$\mathbf{R}^{\theta}_{\mathbf{xy}} = rot(\mathbf{R_{xy}}, \theta) \tag{10}$$

3. For each colour plane multiply the light representation with the reflection map element by element and sum the elements to three scalar values.

$$C(\psi) = \sum_{x=1}^{M} \sum_{y=1}^{M} (R^{\theta}_{xy}(:,:,\psi) \cdot L_{xy}(:,:,\psi)). \tag{11}$$

Efficient algorithm The algorithm presented in the previous section can be made more efficient by pre-computing the eigennumbers. This approach is described in more detail in the second row of Figure 4. One pixel is generated at a time.

To simplify the calculations the eigen-numbers, $\chi(k, \theta)$, are introduced. These are precomputed summations of the multi-

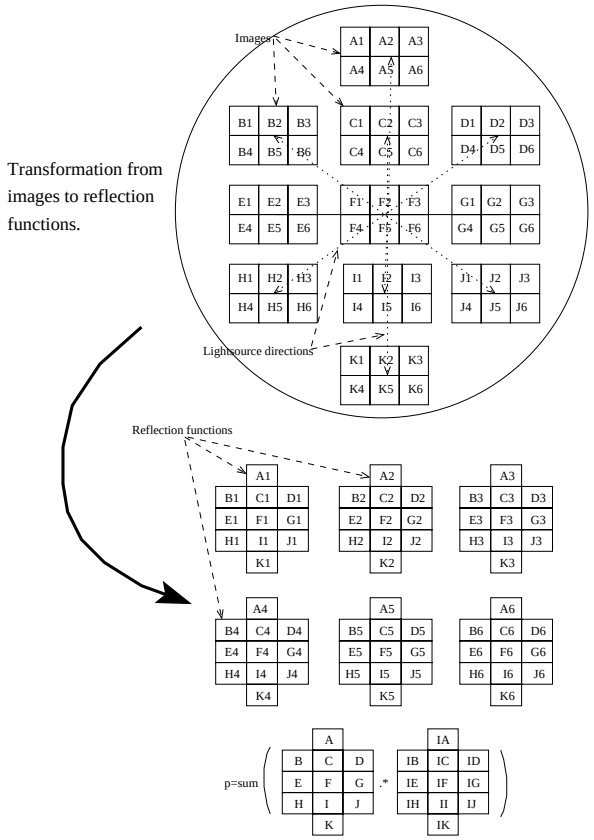


Figure 3: The scene is illuminated with a point light source located in the direction defined by the lightmap. In this figure 11 images with 6 pixels each are captured at varying locations of the lightmap. The pixel-information is then transformed to the reflection functions containing 11 pixels each, which characterize the reflection properties for a specific pixel. In the synthesizing phase the reflection function for a specific pixel is multiplied element by element with the lightmap. The resulting pixel value is equal to the sum of all elements.

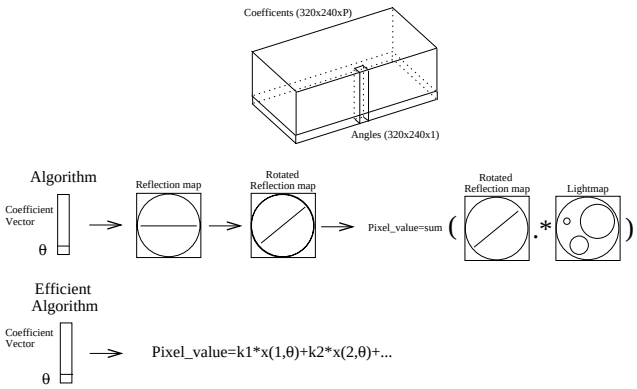


Figure 4: This figure explains the method. The data is represented in a volume, in which each column is used to construct one reflection function. In the first row the algorithm is explained and the second row is a description of the optimized algorithm.

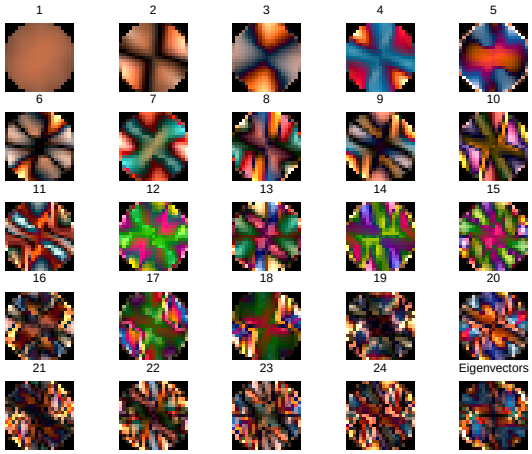


Figure 5: These are the 25 most important eigencomponents for the examined data set.

plications between the rotated eigen-components and the reflection functions

$$\chi(k,\theta) = \sum_{x=1}^{M}\sum_{y=1}^{M} \phi(\mathbf{k},\theta)\cdot\mathbf{R_{xy}}, k = 1:1:n, \theta = 0:\sigma:180. \tag{12}$$

- ψ is the pixel number.

$$\mathbf{R_{xy}}(\psi) = \sum_{k=1}^{N} \kappa(\psi,\mathbf{k})\cdot\phi(\mathbf{k},\theta) \Rightarrow \tag{13}$$

$$\overline{C(\psi)} = (\mathbf{R_{xy}}\cdot\mathbf{L_{xy}}) =$$
$$([\kappa(\mathbf{1},\psi)\cdot\phi(\mathbf{1},\theta)+\ldots+\kappa(\mathbf{P},\psi)\cdot\phi(\mathbf{P},\theta)]\cdot\mathbf{L_{xy}}) = \tag{14}$$

$$\kappa(\mathbf{1},\psi)\cdot[\phi(\mathbf{1},\theta)\cdot\mathbf{R_{xy}}]+\ldots+\kappa(\mathbf{P},\psi)\cdot[\phi(\mathbf{P},\theta)\cdot\mathbf{L_{xy}}] = \tag{15}$$

$$\kappa(\mathbf{1},\psi)\cdot\chi(1,\theta)+\ldots+\kappa(\mathbf{P},\psi)\cdot\chi(P,\theta)$$

- To generate one pixel value (P-1) additions and P multiplications are needed. The resulting equation will for each pixel and colour plane be:

$$C(\psi) = \sum_{k=1}^{P} \kappa(\mathbf{k},\psi)\cdot\chi(\mathbf{k},\theta) \tag{16}$$

3 Implementation and results

There are two main components to the system, the pre-computing phase, in which the data matrices are computed and the synthesizing, where images are generated using novel lighting conditions.

The first phase was implemented in Matlab and the second in C++. An image data set containing 374 images was generated by using MegaPOV (A version of POVray [Povray] allowing rendering of HDR images) and this data set was used to evaluate the

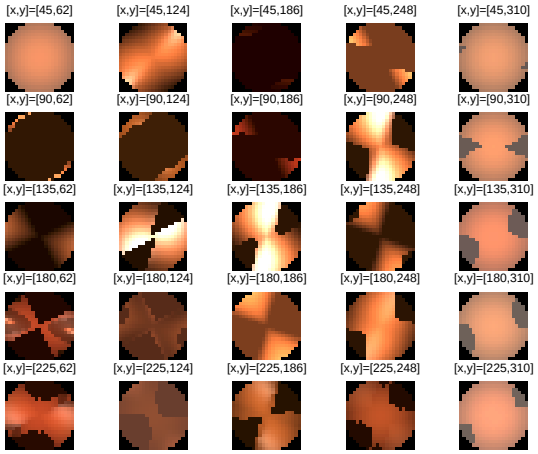


Figure 6: These images are examples of original reflection maps for the data set.

method. The data set contained images of a complicated scene, with a large amount of shadows. In this initial setup small images of size 320x240 pixels were used. Examples of input images can be seen in Figure 2. This dataset was rearranged to reflection functions (See Figure 6) of size 21x21 pixels, which were rotated along their major axes. The 25 most important eigencomponents (See Figure 5) were computed from the rotated reflection functions. It was chosen to compute the 25 most important eigencomponents from the rotated reflection functions, since the use of additional components resulted in little difference in the resulting images. All rotated reflection functions were compressed using the eigencomponents and the coefficients were stored in a data volume together with the rotation angle. To speed up the computations rotated versions of the eigencomponents were precomputed for each of the 360 degrees. The synthesis of an image was divided into two phases. In the first phase lighting coefficients were computed by multiplying the lightmap with the eigencomponents. (See the algorithm in section 2.4). In the second phase the pixel values were computed by using the lighting coefficients and the eigen coefficients. These computations were repeated for every frame.

Resulting images were computed for three well-known HDR fisheye images (See [Debevec and Malik 1997] for more information). These HDR fisheye images were transformed to the same representation as the reflection functions (The upper semi-sphere was used and this was rescaled to 21x21 pixels). For every lightprobe image the corresponding scene was generated using 10 and 25 coefficients and by using the corresponding uncompressed reflection functions (Images generated using 25 coefficients can be seen in Figure 1 and images generated using 10 coefficients and by using the corresponding original lightprobe images can be seen in Figure 7).

The frame-rate for varying numbers of coefficients can be seen in Table 1. As a measure of the image quality the peak signal-to-noise ratio was selected

$$PSNR = 20 * log_{10}(\frac{MAX_I}{\sqrt{MSE}}), \tag{17}$$

where

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \sum_{k=1}^{3} ||im_{orig}(i,j,k) - im_{\kappa}(i,j,k)||^2 \tag{18}$$

In these equations $m \cdot n$ is the size of the image. The reference image generated using the original reflection functions is im_{orig} and an image generated using κ coefficients is im_{κ}. MAX_I is the maximum intensity of im_{orig}.

κ	fps	$PSNR_{lp1}$	$PSNR_{lp2}$	$PSNR_{lp3}$
1	11.25	19.1	20.7	20.0
2	10	18.9	21.6	18.5
3	7.8	18.8	21.7	18.3
5	5.45	19.0	25.5	18.8
10	3.46	19.0	25.4	18.7
25	1.6	19.0	25.6	18.6

Table 1: A comparison of the resulting speed and image quality for various selections of coefficient numbers. The first column is the number of coefficients used, the second is the number of frames per second for the corresponding number of coefficients and the three following columns include the PSNR value for each of the lightprobes.

The errors for different choices of κ were computed for the selections of lightprobe images depicted in the third column of Figure 7. In Table 1 it can be seen that the PSNR increases for increasing numbers of coefficients for the second lightprobe, but for the others the PSNR is approximately constant. The difference in PSNR for 10 and 25 coefficients is very small. However, when examining the resulting images (Figure 1 and 7) it can be seen that the details of some of the spheres become better, when using additional components even if the PSNR value does not improve.

4 Discussion

In this paper we have presented a new approach for real-time relighting of a static scene. This method is general and can be used to relight a scene with arbitrary lightprobe images. It could for example be used to visualize a scene at arbitrary weather conditions by using synthetic sky images (See for example [Olsson et al. 2004] or [Olsson 2005]). The approach could be advantageous for example in scenes with a limited number of materials. The results show that it is feasible to represent BRDFs using principal components. For the data set used in this work 25 coefficients are sufficient. This can be seen by comparing the images in Figures 1 and 7. The images in Figure 1 are very similar to the images in the middle column in Figure 7, which were computed using uncompressed reflection functions. The major limitation is that semi-shadows are not always rendered correctly. The relatively small reflection functions of 21x21 pixels limit the possible shadows, but the limitation can be removed by using larger matrices to represent the reflection functions. A limitation of the method is that it demands detailed reflection functions to compute sharp components. The reflection functions are rotated along their major axes in order to to take the correlation between reflection functions into account.

This is a simple approach and can not directly be compared with more general, but also more complicated approaches as for example the method presented in [Kristensen et al. 2005]. The goal of this method is to relight a still image in novel lighting conditions, while the goal in their paper is to relight an arbitrary scene with unstructured light clouds using precomputed radiance transfer. The major advantage of this method is that each pixel of the resulting image can be computed with a fixed number of 25 multiplications and 24 additions, while the more advanced algorithms need a much larger number of operations in the generation process. On the other hand, the largest drawback is the precomputing step and the relatively large data matrices, which must be stored to be able to gen-

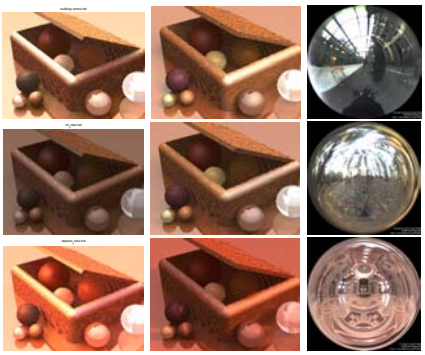


Figure 7: Each row of images corresponds to a specific lighting condition defined by an associated light probe image. The first image was generated using 10 coefficients, the second was generated using the original reflection functions and in the third image the corresponding lightprobe image can be seen.

erate images. The memory usage is

$$x_{size} * y_{size} * 3 * (Nbr_{coefficients} + 1), \quad (19)$$

which means that for an image of size 1024 x 1024 pixels, as much as 76 megabytes is needed to store the information.

The memory demand increases for larger choices of reflection functions, but the number of coefficients do not increase at the same rate, which means that after the precomputing step it should be possible to use much larger reflection functions and thus increase the image quality, without severely increasing the memory usage of the data matrices.

This method should be most appropriate for small images to make it possible to easily generate an image for various lighting conditions, for example the appearance of a car during the day.

5 Future Work

As a major limitation of this implementation is the computing time, our major efforts will be directed in decreasing the execution time. This can be performed by implementing parts of the algorithm in hardware. See for example [Owens et al.]. Another future work is to generalize the algorithm to 3D, by generating multiple views.

References

BLINN, J. F., AND NEWELL, M. E. 1976. Texture and reflection in computer generated images. In *Communications of the ACM*, vol. 19, 542–547.

DEBEVEC, P. E., AND MALIK, J. 1997. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH*, 369–378.

EPSTEIN, R., HALLINAN, P. W., AND YUILLE, A. L. 1995. 5±2 eigenimages suffice: An empirical investigation of low-dimensional lighting models. In *IEEE Workshop on physics-based vision*, 108–116.

GREENE, N. 1986. Environment mapping and other publications of world projections. In *IEEE Computer Graphics and Applications*, vol. 6.

HAYKIN, S. 1999. *Neural Networks*. Prentice Hall, New Jersey.

HO, P.-M., WONG, T.-T., AND LEUNG, C.-S. 2003. Compressing the illumination-adjustable images with principal component analysis. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, 59–64.

JOLIFFE, I. T. 2002. *Principal Component Analysis*. Springer-Verlag, New York.

KRISTENSEN, A. W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Precomputed radiance transfer for real-time lighting design. In *Proceedings of ACM SIGGRAPH*. 1208–1215.

LEHTINEN, J., AND KAUTZ, J. 2003. Matrix radiance transfer. In *Proceedings of the 2003 symposium on interactive 3D graphics table of contents*, 59–64.

LIU, X., SLOAN, P.-P., SHUM, H.-Y., AND SNYDER, J. 2004. All-frequency precomputed radiance transfer for glossy objects. In *Eurographics Symposium on Rendering*.

MATUSIK, W., PFISTER, H., ZIEGLER, R., NGAN, A., AND MCMILLAN, L. 2002. Acquisition and rendering of transparent refractive objects. In *Proceedings of Eurographics Workshop on Rendering*, 267–278.

NIMEROFF, J. S., SIMONCELLI, E., AND DORSEY, J. 1994. Efficient re-rendering of naturally illuminated environments. In *Eurographics workshop on rendering*.

NISHINO, K., SATO, Y., AND IKEUCHI, K. 1999. Eigen-texture method: Appearance compression based on 3D model. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 618–624.

OLSSON, B., YNNERMAN, A., AND LENZ, R. 2004. Visualizing weather with synthetic high dynamic range images. In *Proceedings of SPIE - IS & T Electronic Imaging. Visualization and Data Analysis*, R. F. Erbacher, J. C. Roberts, M. T. Gröhn, and K. Börner, Eds.

OLSSON, B. 2005. *Image based visualization methods for meteorological data*. Licentiate thesis, Department of Science and Technology, Linköping University. ISBN 9185297003.

OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRUGER, J., LEFOHN, A. E., AND PURCELL, T. J. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, 21–51.

POVRAY. Persistence of vision raytracer (version 3.6). *Computer software, Retrieved from http://www.povray.org/download/*.

SHIM, K. H., AND CHEN, T. 2005. A statistical framework for image-based relighting. In *IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency light environments. In *Proceedings of the 29th annual conference on computer graphics and interactive techniques*. 527–536.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. In *ACM SIGGRAPH*. 382–391.

WOOD, D. N., AZUMA, D. I., ALDINGER, K., CURLESS, B., DUCHAMP, T., SALSIN, D. H., AND STUETZLE, W. 2000. Surface light fields for 3d photography. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques*. 287–296.

Opportunities and challenges when 3D accelerating mobile user interfaces

Mikael Persson
e-mail: mikael@tat.se
TAT AB

Karl-Anders Johansson
e-mail: k-a@tat.se
TAT AB

Abstract

This paper addresses new techniques and challenges for user interface design for small-screen devices made possible by the recent availability of 3D graphics hardware. A survey of current research on the topic for desktop systems is presented and applications of these techniques for small screen devices are proposed. Also general differences in user interface design between large and small screen devices are highlighted.

Keywords: Graphics hardware, user interfaces, mobile devices

1. Introduction

Supporting graphics hardware in the user interface is becoming increasingly more popular for desktop systems, latest versions of Mac OS X and Microsoft Windows will employ 3D hardware for a range of new user interface techniques and effects. Currently we are seeing an explosion in graphics performance for both 2D and 3D on handheld devices made possible by the introduction of graphics hardware. This explosion is mainly driven by gaming but may be exploited for enabling a better user experience. Designing user interfaces for small screen devices is a tedious task mainly focused at organizing screen content such that large data sets can be presented in an intuitive way and to provide an interaction hierarchy that is easy to understand and use. The introduction of 3D graphics hardware makes a new range of techniques available and feasible on mobile devices that may help with providing more appealing, enjoyable and intuitive user interfaces. This paper aims to present previously described technical solutions for accelerating the user interface with graphics hardware on desktop systems and describe what challenges that must be overcome in order to realize these techniques on mobile devices with current and future graphics hardware. We also intend to highlight some interesting user interface design techniques found in the desktop space that may be applicable to mobile devices with graphics hardware.

In section 2 a survey of how 3D acceleration is currently being used in user interfaces as well as an overview on current research on the topic for desktop systems is presented. Section 3 contains a summary of technical limitations and challenges encountered when attempting to implement similar systems on mobile platforms. Finally in section 4 we outline aspects of user interface design for small screen devices related to the techniques described in section 2 as well as suggestions for new research.

2. Survey of 3D acceleration in desktop UIs

The dominant user interface model for desktop systems is window management. In [Myers 1988], Myers defines a window manager as "*a software package that helps the user monitor and control different contexts by separating them physically onto different parts of one or more display screens*". He adds: *"Before window managers, people had to remember their various activities and how to switch back and forth*". Generally speaking the window manager controls the physical display area and manages the different user interface components and applications that aspire on drawing graphics. Tasks that fall on the window manager are typically overdraw management, moving and resizing windows and controls as well as managing pointer devices such as a mouse. Some current systems [Graffagnino 2002; McCartney 2002] target this piece of software for 3D acceleration.
Traditionally user interface graphics systems draw directly to the display frame buffer via the window manager. The window manager controls which areas of the display are owned by a given application. If a certain area of the display need to be updated the window manager orders the application to draw its content directly to the display in the given area. Exploiting graphics

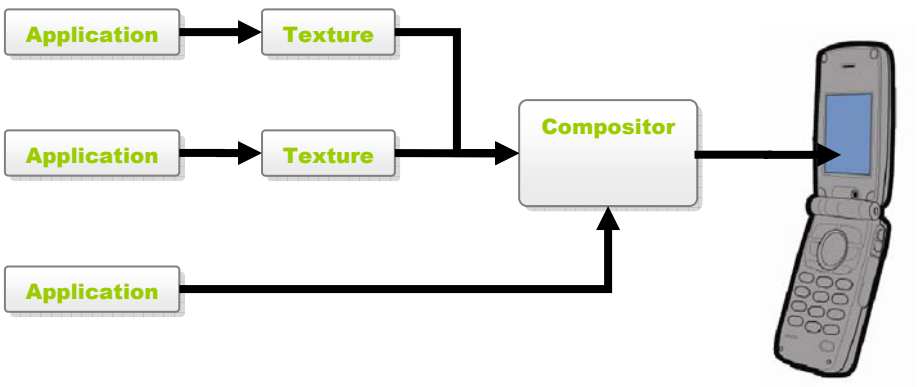


Figure 1. A common approach to accelerating a traditional window manager is the compositing approach, illustrated above.

acceleration without breaking compatibility with existing applications is generally achieved by introducing a compositor that manages the display instead of the individual application via the window manager, see figure 1. This is a common approach that is available in current generation Apple OS [Graffagnino 2002] and will be available in next generation of Microsoft OS [McCartney 2002]. In this approach the application draws its user interface to a texture accessible by the graphics hardware using existing methods. The different textures containing the application UIs are composited using the graphics hardware. Furthermore the compositor may use the graphics hardware to apply a wide range of effects during compositing, such as per-pixel transparency, pre-window fade and transform control, independent of the original application user interface code. In conjunction applications specifically written for the compositing pipeline may use the graphics hardware directly to further enhance the user interface. The compositor generally access the graphics hardware via an open API such as OpenGL [Graffagnino 2002; Kawahara and Byrne 2005] or a proprietary such as DirectX [McCartney 2002].

With a system as described above, a range of different new user interface concepts become available. A popular use case is organising windows on the desktop. In [Kawahara and Byrne 2005; Robertson et al. 2000; Rousel 2003] windows may be stacked at the side of the screen by rotating them in 3D, see figure 2. This gives the user overview of the content of the windows while freeing up space for more applications. Given a high performance multi-tasking system the content of the windows may be updated continuously such that the user can monitor any activity while enjoying the space for other applications. Other solutions to this problem include the *exposé* feature of Mac OS X Tiger. Instead of stacking the windows using perspective transforms the *exposé* feature tiles all of the open windows - scales them down and arranges them, so that all are completely visible. This allows the user to get an overview of all open windows. It also allows the user to select one of the miniature windows which will bring that window to front, this feature makes *exposé* useful both to get an overview of current running applications as well as application switching. Both features would not be possible without seriously draining system resources if not graphics hardware were employed.

Some researchers propose designs based on a compositor WM approach that are more decoupled from the traditional desktop metaphor UIs. G. Robertson et al. have presented a solution called *The Task Gallery* [Robertson et al. 2000] where user tasks (i.e. applications windows) appear as artwork hung on the walls of a virtual art gallery. The Task Gallery aim to exploit the spatial cognition and memory inherit in humans and is based on the theory that if presented with a virtual environment that is more like the 3D environment that we live in new and old users will find interaction more intuitive and enjoyable. According to the authors user studies have shown that the Task Gallery helps with task management in the way that users easier remember where they "put" their windows in the 3D metaphor. Although this concept is far away from the traditional desktop metaphor many designs that have found their way into current commercial systems such as stacking windows with perspective transforms are presented in this paper.

Interesting research projects are *Project Looking Glass* [Kawahara and Byrne 2005] and *Ametista* [Rousel 2003] two open source window mangers that are based on the 3D interaction metaphor. These projects aim to provide a platform for research on 3D user interfaces in a real working window environment. Perhaps the most interesting project is [Kawahara and Byrne 2005] where

Figure 2. Project Looking Glass among others uses 3D perspective transformations to stack windows at the side of the screen to free up space for other applications. (Image courtesy of Project Looking Glass.)

Figure 3. Project Looking Glass includes a media player with a free form 3D user interface. Using 3D many new and interesting UI techniques become available. (Image courtesy of Project Looking Glass.)

Figure 4. Project Looking Glass also includes a photo browser with a free form 3D user interface. (Image courtesy of Project Looking Glass.)

Java is used as the platform for building applications that expose 3D user interfaces. The window manager has a number of built in interaction paradigms such as stacking windows, task bars and how navigation is performed in the 3D space that the applications occupy. Besides from that novel interaction methods may be introduced by the individual projects. Some interesting applications have been developed as part of the project, most notably a media player and a photo browser. These applications have free form 3D user interface floating in the application space as oppose to traditional applications that are contained within their windows, see Figure 3 and Figure 4.

Another popular field of user interface research that doesn't require hardware acceleration but that definitely benefits from it is *zoomable user interfaces* or ZUI. Both OpenGL [Blythe and Munshi 2004] and the emerging standard OpenVG [Rice 2005] are based on hierarchical transformations which combined with more graphics performance via hardware acceleration makes a great platform for ZUI. The goals of ZUI is to better visualize large data sets on small display areas. The capability and performance to seamlessly change level of zoom enables a range of interesting user interface techniques. Photomesa [Graffagnino 2002] is an example of a browser that uses novel layout mechanisms (quantum treemaps and bubble maps) that allows users to see as many photos as possible and maintain context. It allows users to group photographs by date, filename and directory as well as employ zooming techniques to display the photographs. The basic concept is fairly simple, photos are clustered with regard to a number of parameters such as data, texture, color etc and ZUI is used to let the user seamlessly navigate the collection of images. The user directly control the level of zoom and the software attempts to always present the level of information that is suitable for the given level of zoom. It has been shown [Cockburn and Savage 2003] that techniques where the user directly controls the degree of zoom does not aid in searching for images, generally when a new degree of freedom for navigation is introduced the learning curve for the UI is steeper. An interesting extension of ZUI to remedy this problem is *speed dependent automatic zoom* (SDAZ). This concept is based on the fairly simple assumption that when the user is actively searching for an image a thumbnail overview is more suitable and when the user settles down on a given image a more zoomed in view is wanted. A photo browser based on the SDAZ concept is described in [Igarashi and Hinckley 2000; Cockburn and Savage 2003], and according to the authors user studies show that this technique aid some users in searching for photos.

3. Technical limitations and challenges

The introduction of graphics hardware for mobile devices presents great opportunities for building more intuitive and overall better user interfaces. However, when attempting to facilitate the research presented in section 2 for mobile devices a number of challenges arise, some of these challenges and limitations are covered in this section.

Lately, good standards for accessing the hardware such as OpenGL|ES and OpenVG have emerged. OpenVG is a much newer standard, and there are no hardware implementations yet. This section will focus on limitations of OpenGL|ES 1.x as this has been tried in the industry. Most OpenGL|ES graphics accelerators for the mobile market today offer quite good raw rendering performance and give major speed improvements on pure blitting and alpha blending. OpenGL|ES provides a convenient and standardized way of performing controlled composition of bitmaps and is capable of supporting at least half of the Porter-Duff blending operations [Porter and Duff 1984]. Details on blending limitation follow below. Current graphics hardware is capable of performing full screen bitmap composition at rather impressive frame rates as would be expected since the most important target software is games. Using the hardware graphics accelerator in a window system would mean enabling many of the features of a compositing based window system as described in section 2 without putting much load on the CPU. When using OpenGL|ES for this purpose, window content needs to be accessed as texture data. Most of today's OpenGL|ES implementations require textures to be uploaded to dedicated memory which presents a number of technical challenges that must be addressed:

Small texture memory
The typical OpenGL|ES accelerator on the market today targeting QVGA displays has about 1MB of VRAM (video ram) to be shared by frame buffer and textures. This clearly indicates that the window system must contain some sort of VRAM virtualization approach in order to fit all windows in "virtual VRAM". This is a problem that is common with desktop solutions [Graffagnino 2002; McCartney 2002]. By treating the VRAM as an on chip "L1" cache, it is possible to page in textures from a system RAM "L2" cache. Another issue is if the device has enough RAM to hold the "L2" cache. If not, the window system must resort to issuing a repaint command to the client in order to get the bitmap data. If this happens for every frame, we will not gain any performance boost from using the hardware graphics accelerator. More likely, the added overhead will in fact make the UI slower than if not using any hardware acceleration.

Slow texture transfer
In order to save power, all bus widths and -speeds are generally kept thin and slow on mobile devices. Also, graphics hardware manufacturers do not typically prioritize optimizations of the particular data path for transferring texture data since most games and benchmarks focus on fill rate and polygon count. This definitely has an impact on the VRAM virtualization mentioned above since it relies on being able to quickly swap in textures from RAM. The general solution to this problem is texture compression which lowers the impact on narrow busses but we will see below that this is not always feasible for user interfaces. The best workaround for the slow texture transfer is to reduce the number of client side updates and instead rely on effects and animations that are possible to perform on the graphics accelerator. Examples of this are scaling, moving, and opacity changes.

UI graphics is not suitable for texture compression
User interface graphics such as text and fine lines are very sensitive to compression artifacts. This effectively rules out texture compression for these tasks. Also, since applications and window content is rendered on the device in real time, texture compression times would pose a problem since it further delays the uploading of the texture data. Not being able to use texture compression further increases the texture memory problems (size and speed) mentioned above.

Texture size
OpenGL|ES 1.x supports neither texture sizes larger than 256x256 nor sizes other than powers of two. This means that texture RAM can not be utilized to 100% efficiency since windows rarely have sizes that are powers of two. It is possible to use tiling to work around this problem, in which case source bitmaps are broken down into small tiles that can be allocated from larger textures. The main problem with this is the vastly increased complexity of

the drawing operations while maintaining good performace. There is also a significant increase in the geometry complexity which may also cause performance degradations.

Blending limitations on OpenGL|ES 1.x
As mentioned above, only half of the 12 Porter-Duff blending modes are supported by OpenGL|ES 1.x. The reason for this is that in OpenGL|ES1.x there is no support for destination alpha. The ones that remain are "Clear", "Src", "SrcOver", "DstIn", "DstOut" and "Dst". However, there is support for additional blending modes, for instance a version of "SrcOver" without the requirement of RGB being premultiplied with alpha

OpenGL|ES and other concurrent hardware
On a mobile device there is likely to be special hardware for decoding video and, if applicable, displaying a camera viewfinder. This can conflict with OpenGL|ES operation. Normally OpenGL|ES would not be running when using these special hardware features but with a hardware accelerated window system, OpenGL|ES is running all the time. Note that this differs very much between different OpenGL|ES implementations and integration decisions. For instance, an OpenGL|ES accelerator may include an interface for a camera and require that viewfinder- and snapshot data is stored in the chip's embedded memory. This may end up consuming most of the accelerator's RAM, not leaving any room for the window "L1" cache mentioned above, thus forcing the window system to revert to non-accelerated mode. Another example could be a graphics accelerator that actually does not at all support running the OpenGL|ES core when the camera viewfinder is active even if the RAM issue was solved.

4. UI design challenges and possibilities

In general when working with small-screen devices only one window/application may be simultaneously visible which has brought us back to the point where it is difficult to remember and organize activities. Generally for small-screen devices as opposed to desktop systems the concept window is analogous with screen, i.e. the window occupies the entire screen. User interaction is layed out in a hierarchical fashion, selecting an option on one screen generally presents another screen with more options, pressing the back key returns to the first screen. In these systems user interface designers often struggle with keeping the context such that the user is always aware of where a certain choice will take him/her and where the back key will lead at any given time, which as the complexity of the system grows becomes an enormous task.

Another major challenge and difference when designing user interfaces for mobile devices versus desktop systems is that on a mobile device you are most likely targeting first time users. On a desktop system you are designing a tool and optimizing it for maximum efficiency, on a mobile device on the other hand you are designing a system that has to be intuitive enough to be usable by first time users. Generally mobile devices do not come with a thick user manual and anyway you are expected to be able to use the device without reading any more instructions than what is presented on the display in the user interface.

Mobile 3D graphics presents many interesting opportunities for improving the problem with context shifts and its implications on usability as outlined above. Different kinds of transitions are very effective in helping the user to remember which context is currently active.
With the performance of graphics hardware and 3D a wide range of new transitions are available and a lot of research is required to evaluate these transitions and how they may aid the user. One example of new transitions that may help the user is presented in figure 5, by connecting context menus associated with an object in a visually appealing way the human sense for spatial context may make these transitions more intuitive.

Figure 5. By connecting context menus associated with an object in a visually appealing way the human sense for spatial context may make these transitions more intuitive.

Both graphics and general processing performance is increasing at a rapid rate however display sizes are still limited by physical constraints such as the fact that the device must be able to fit in the pocket or in the palm of your hand. The introduction of high performance 3D graphics may be exploited to virtually extend the size of the screen by introducing new techniques for better organizing screen content. Section 2 discussed some concepts on desktop systems that attempt to solve this problem. This section will further discuss these topics in the context of mobile devices.

Stacking is a very common user interface technique used to display a set of components, icons, windows or pages in a compact manner. The components appear to be stacked or piled on top of each other with only a portion or tab visible. This tab is used to switch between the current active component and the component indicated by the tab, this is commonly referred to as tabbed windows introduced in [Beaudouin-Lafon 2000] and extended in [Beaudouin-Lafon 2001]. Stacking is especially attractive for small screen devices since the technique allows for quick navigation among pages on a limited space, where otherwise each page had to be its own window. However, a common problem of packing information tightly is that it will most likely be less intuitive. In such cases, visual perception will become critical for the user to understand what is displayed.
A user study [Kjelldahl 2003] has shown that the visual cues perspective and shadow have a substantial positive effect on perceiving position. Figure 6 shows two examples of stacking images (for instance in a photo viewer application): One without the perspective and shadow and one where the two visual cues have been used to help the user perceive the scene.
With the use of modern graphics hardware and the compositor approach, effects such as perspective transforms (texture mapping) and shadow techniques may be used on arbitrary user interface components to achieve the right visual cues and thus reduce the risk of the user misinterpreting the intended use.
As described in section 2 recent products and research on desktop systems that employ graphics hardware in the user interface

Figure 6. Normally tabbing is achieved by simply displacing the images (bottom). Using visual cues such as perspective and shadows the UI may be made more intuitive (top).

Figure 7. Another example of where stacking in perspective can be used. Here used for application switching.

attempt to achieve better organization of screen content by stacking windows at the side of the screen with perspective transformations thus exploiting these visual cues. Although stacking windows by the side of the display would be feasible on a mobile device, the small screen size would most likely render the window content impossible to see. The same unfortunately apply for the *exposé* feature. Unless only two or three applications are running, scaling down all open windows such that they all fit the display is not feasible for most mobile devices due to the small screen. Instead stacking combined with the visual cues perspective and shadow may be used to give an intuitive overview on the limited display space, see figure 7. This effect may be useful for more intuitive application switching or simply to get an overview of currently open windows.

Improving screen content organization and virtually extending the display on small screen devices is one of the biggest challenges for user interfaces designers and much research is needed. The recent introductions of OpenGL|ES and OpenVG plus hardware acceleration will, as mentioned in section 2, provide an excellent platform for *zoomable user interfaces*. Zoomable user interfaces is a popular research field for both mobile and desktop systems with the goal of better visualizing large data sets on small displays. As storage capabilities of mobile devices and available information via network connections increase rapidly, the need for new and better ways to visualize this data on small screen devices are imperative. ZUI have great potential for improving screen content organization in a natural way. The PhotoMesa [Bederson 2001], also mentioned in section 2, has also been developed in a version for PocketPC [Khella and Bederson 2003]. As mentioned in section 2, problems were encountered when introducing new navigation methods, i.e. direct controls for zooming. The lack of sophisticated input devices such as a mouse/point on many mobile devices is likely to make it even less attractive to introduce new navigational degrees of freedom. This makes methods such as SDAZ very interesting. Interesting research would be to extend the concept of SDAZ to work well with navigation keys and other systems that do not have pointer devices.

5. Conclusions

High performance 3D graphics hardware will enter the mobile arena driven by the gaming industry and thus it is very likely that the mobile user interface will evolve in the same direction as the desktop systems. Some of the new user interface techniques developed for these systems may provide great solutions for problems inherited in mobile devices such as better screen content organization on small screens via *zoomable user interfaces* or more intuitive interaction flows via transitions. More research is needed to adapt, evolve and evaluate these techniques for small screen limited mobile devices.

References

BEAUDOUIN-LAFON, M. 2001. Novel interaction techniques for overlapping windows. In Proceedings of ACM Symposium on User Interface Software and Technology, UIST 2001, Orlando (USA), ACM Press. Pages 153-154.

BEAUDOUIN-LAFON, M. and LASSEN, H.M. 2000. The architecture and implementation of CPN2000, a post-WIMP graphical application, in UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters 2(2). Pages 181- 190

BEDERSON, B. 2001. Photomesa: A zoombable image browser using quantum treemaps and bubblemaps, in Proceedings UIST 2001, ACM Press. Pages 71-80

BLYTHE, D. and MUNSHI, A. 2004. OpenGL|ES Common/Common-Lite Profile Specification Version 1.1.04 (Annotated), www.khronos.org

COCKBURN, A. and SAVAGE, J. 2003. Comparing Speed-Dependent Automatic Zooming with Tradional Scroll, Pan, and Zoom Methods, People and Computers XVII: British Computer Society Conference on Human Computer Interaction. Bath, England. Pages 87-102.

GRAFFAGNINO, P. 2002. Apple OpenGL and Quartz Extreme. Presentation at SIGGRAPH 2002, OpenGL BOF.

IGARASHI, T. and HINCKLEY, K. 2000. Speed-dependent automatic zooming for browsing large documents, Proc. UIST 2000, ACM Press. Pages 139-148

KAWAHARA, H. and BYRNE P 2005. Project Looking Glass. Presentation at Sun JavaOne 2005. https://lg3d.dev.java.net/

KHELLA, A. and BEDERSON, B. 2003. A Zooming Image Browser for the Pocket PC. University of Maryland, Technical Report

KJELLDAHL, L. 2003. A survey of some perceptual features for computer graphics and visualization. SIGRAD2003

MCCARTNEY, C. 2002. Windows Desktop Composition. Presentation at WinHEC 2002, Windows Graphics Architecture session.

MYERS, B.A. 1988. A taxonomy of window manager user interfaces. IEEE Computer Graphics and Applications, 8(5). Pages 65-84

PORTER, T. and DUFF, T. 1984. Compositing digital images. Computer Graphics (SIGGRAPH '84 Proceedings), vol. 18, no. 3, pages 253-259

RICE, D. 2005. OpenVG 1.0 Specification. www.khronos.org

ROBERTSON, G., VAN DANTZICH, M., ROBBINS, D., CZERWINSKI, M., HINCKLEY, K., RISDEN, K., THIEL, D. and GOROKHOVSKY, V. 2000. The Task Gallery: a 3D window manager. In Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems, ACM Press. Pages 494-501.

ROUSSEL, N. 2003. Ametista: a mini-toolkit for exploring new window management techniques. In Proceedings of CLIHC 2003, Latin American Conference on Human-Computer Interaction, ACM Press. Pages 117-124.

Distributed Fractal Generation Across a Piconet

Daniel C Doolan, Sabin Tabirca*
University College Cork,
Ireland.

Abstract

This paper explores the distributed generation of fractal images on mobile phones using Bluetooth (JSR-82) as the inter-device communications mechanism. Mobile phones inherently have limited resources (such as memory and processing power). The aim of this paper is to demonstrate a technique, where the resources of several mobile devices can be used in conjunction. This will allow for the computation of a processor intensive task in a short period of time.

CR Categories: I3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Geometric algorithms, languages and systems.; D1.7 [Programming Techniques]: Visual Programming; C2.4 [Computer Communication Networks]: Distributed Systems—Client/Server.

Keywords: Fractal Image Generation, Distributed Computing, Bluetooth, Piconet

1 Introduction

Fractals have been a classical topic ever since what is considered to be the discovery of the first fractal in 1872 by Georg Cantor, (typically called "Cantor Dust"). The present interest in Fractal geometry is mainly due to the work of Benoit Mandelbrot, who in the mid 1970's was successful in the creation of the an index (Mandelbrot Set) for all the possible Julia Sets. Since then their have been many further developments such as Fractal Terrain Generation [Meyer 1982] [Lucas and Marquand 1983], Fractal Image Compression [Lu 1997], Fractal Encryption and Fractal Music. However, the process of generating such images requires significant computational resources [Doolan and Tabirca 2005]. In this paper we present a study parallelizing the generation of Mandelbrot images on mobile phones using Bluetooth as the communication mechanism.

Over the past couple of years both Sony-Ericsson and Nokia Developer websites have published invaluable training material on the development of J2ME applications with Bluetooth technology.

Some interesting literature has been published of late, dealing with Bluetooth technology. One thesis [Long 2004] "A Study of Java Games in Bluetooth Wireless Networks" gives an overview of the Bluetooth API and shows how such technology can be employed in the development of wireless networked games. The thesis gives a simple example of a game designed in a Point to Point Piconet Configuration (Figure 1). It also includes some useful references to various sources dealing with both game programming and networked communications.

*e-mail: {d.doolan, tabirca}@cs.ucc.ie

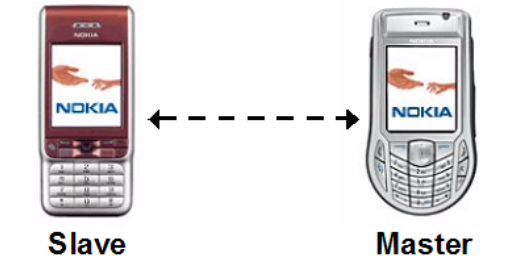


Figure 1: Simple Point to Point Piconet Configuration.

A far more in-depth look at Bluetooth technology with J2ME may be found in the MSc thesis of [Klingsheim 2004]. The thesis centered around the development of two Bluetooth applications and the testing of same on several devices, namely Nokia 6600 to Nokia 6600, Sony Ericsson P900 and PC to Nokia 6600. The applications dealt with device discovery and bench-marking data transfer.

The generation of the Mandelbrot Set is often referred to as an "embarrassingly parallel computation". It can be easily divided into a number of completely independent parts, each of which can be executed by a separate processor. The parallel computation of the Mandelbrot Set usually relies on the C based Message Passing Interface (MPI) library [MPI][Book][O'Mahony 2004].

More and more mobile devices are shipping with Bluetooth technology as standard. Typically all Smartphones fall within this remit, examples include the Nokia Series 60 2^{nd} and 3^{rd} Edition phones. Sony-Ericsson so too have a significant number of Bluetooth enabled devices (K750, D750, W800, K608). The Java capabilities of these devices are that of levels 5 and 6 of Sony-Ericssons Java Platforms [Sony-Ericsson 2005].

2 Fractal Generation

The generation algorithms for the Mandelbrot Set are quite simple, but as the generating function is iterated repeatedly this gives rise to complexity and results in a highly complex fractal image. Fractals are usually obtained when the generating function $f(z)$ is non linear. For $f(z) = z^2 + c$ the classical Mandelbrot Set is obtained. Mandelbrot-like sets are also obtained when the generating function has the form of $f(z) = Z^u + C^v$.

The area of the Fractal image that is displayed is dependent on the xmin,ymin, xmax,ymax values. Figure 2 shows a typical Mandelbrot Image and the corresponding coordinates for it.

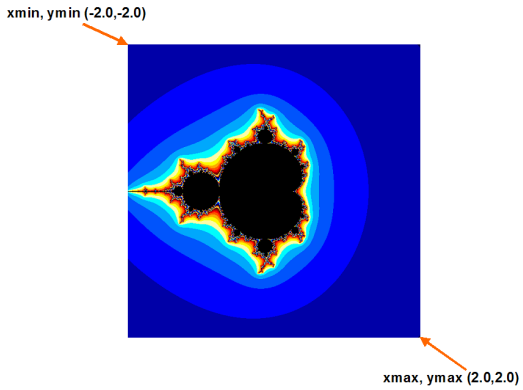


Figure 2: Typical Mandelbrot Image for xmin,ymin = -2.0, xmax,ymax = 2.0.

A Code sample (Listing 1) of the Mandelbrot Set generation function shows a direct relationship with the Mandelbrot Set Algorithm. One can clearly see that if the absolute value of the complex number lies outside the threshold R, that the pixel at the current coordinates of the iteration through the image will be drawn in a specific colour.

```
1  for(int i=0;i<SIZEX;i++)for(int j=0;j<SIZEY;j++){
2      Complex c = new Complex((XMIN+i*STEPX),(YMIN+j*STEPY));
3      Complex z = new Complex();
4      for (k=0;k<NR_ITER;k++){
5          z=f(z,c);
6          if(z.getAbs()>R){
7              r = c[k%l][0]; g = c[k%l][1]; b = c[k%l][2];
8              color = b + (g<<8) + (r<<16) + alpha;
9              pixels[(j*SIZEX) + i] = color;
10             break;
11         }
12     }
13 }
```

Listing 1: Code listing of Mandelbrot Set Function

The execution times for generating a fractal image on a mobile phone are quite long (Table 1), especially as the number of iterations required increases. Execution on a Nokia 6630 Phone requires 55,657ms to generate a 200 x 200 pixel image using 500 iterations, running at 1000 iterations the processing time increases to 98,250ms (almost double). Similar tests using Sun's WTK emulator yields results far in excess of the execution times for the Nokia 6630 Phone. The Sony-Ericsson Emulator demonstrated the fastest computation.

Device	500	750	1000
SE WTK 2.2	30,079 ms	42,516 ms	56,141 ms
Nokia 6630	55,657 ms	75,266 ms	98,250 ms
Sun WTK 2.2	140,875 ms	205,078 ms	269,734 ms

Table 1: Image Generation Times for the Mandelbrot Set (Image Size 200x200 pixels, xmin,ymin -2.0, xmax,ymax 2.0) at varying number of iterations.

3 Distributed Fractal Generation

The application was developed using Sun Java Studio Mobility 6 2004Q3 [Sun-Microsystems a] and the J2ME Wireless Toolkit (WTK) Emulators [Sun-Microsystems b].

The application has been designed to work with several fractal formulas and not just fractals of the form $f(z) = Z^u + C^v$. The other implemented forms include $f(z) = Z^u - C^v$ and $f(z) = Z^u + C^v + Z$. A more comprehensive Math's Class would allow for a wider diversity of possible fractal formulas. The MathFP class is a efficient integer based alternative to this.

The design of the system is that of a Point to Multi-Point configuration (Figure 3). The Point to Multi-Point configuration is significantly more complex than the simpler Point to Point configuration. For the Point to Multi-Point Configuration to work the Master Device must keep a list of all clients connected to it.

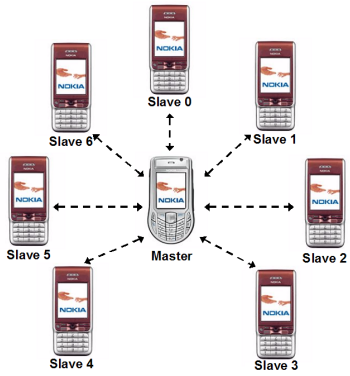


Figure 3: Point to Multi-Point Piconet Configuration.

3.1 User Interface

One of the most important parts of the Server Program is the User Interface for modifying the Fractal Image settings. The most important options include the Image Size, Number of Iterations, Fractal Co-ordinates and Fractal Equation Type (Figure 4). The Client Application has a simple interface that displays whether an image is currently being generated or not, the processing time for the generation of an image and the Image Settings the Client is currently using to generate the image (Figure 4).

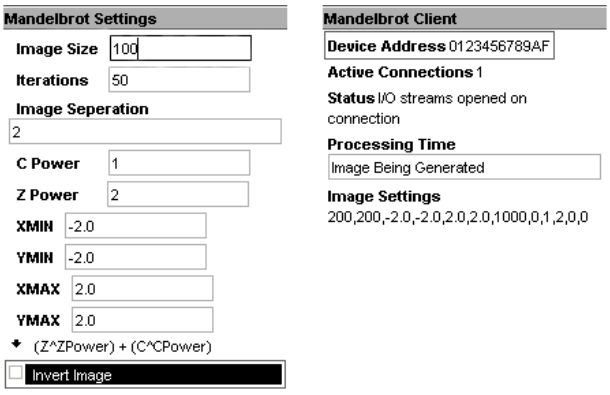


Figure 4: Fractal Image Settings Form, (Server). Client Info Form.

3.2 Work Load Balancing

There are several ways in which the image can be divided among the client's nodes for processing. These include Uniform Block, Cyclic and Dynamic Load Balancing.

3.2.1 Uniform Block Balancing

This method of load balancing was the first scheme to be implemented in the application. In this implementation the image is divided into *nrClients* vertical (or horizontal) strips of equal size (Figure 5). The client i receives for computations the image chunk with the consecutive columns $i \times w/nrClients, i \times w/nrClients + 1, ..., (i+1) \times w/nrClients - 1$. This division of the image into similar vertical strips would yield to similar computation on each client. It is a simple scheme where a single message is sent to each client which results in a corresponding message being returned to the master with the results of the computation. A similar method [Murty 2004] where the image is divided into a grid was used for the generation of Fractal Images using .NET Web Services.

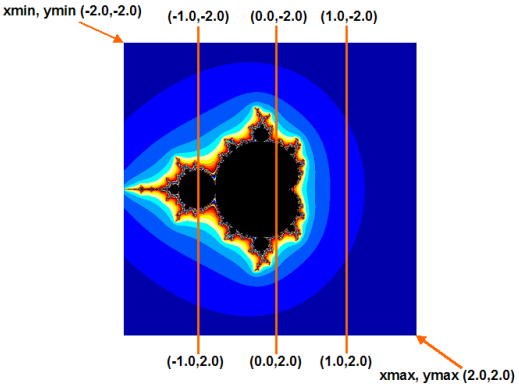


Figure 5: Division of Image into Vertical Strips.

For example dividing the image into four vertical strips yields a matrix of coordinates that may be sent to the clients connected to the Master. In the case of 4 vertical strips with xmin, ymin starting at -2.0 and xmax, ymax starting at 2.0, the following matrix is generated (Table 2). In the case that 4 clients were used using a grid of sub-squares the resulting matrix would center around the value 0.0 for a Mandelbrot Image where xmin, ymin = -2.0 and xmax, ymax = 2.0.

xmin	ymin	xmax	ymax
-2.0	-2.0	-1.0	2.0
-1.0	-2.0	0.0	2.0
0.0	-2.0	1.0	2.0
1.0	-2.0	2.0	2.0

Table 2: Co-ordinate Matrix for 4 Vertical Strips: xmin,ymin = -2.0, xmax,ymax = 2.0

Several other elements are essential for the distributed processing of the image. This size of the image each client is to generate. An identifier for which section of the image was generated by each client. This is necessary so the image can be reassembled on the Server in the correct order. The order in which images are returned to the Server is random as some sections will require greater processing time than others. The other parameters include whether the image should be inverted or not, the powers for Z and C, the number of iterations to be executed and finally a flag for the type of fractal formula to use.

3.2.2 Cyclic Load Balancing

This is a follow on approach from the Uniform Block Load Balancing example. The image is still divided into equal sized vertical strips $\{S_0, S_1, ..., S_{p-1}\}$ each of them containing only a few columns. The partition of this p strips onto the clients is performed into a cyclic matter so that the client i would receive the strips $\{S_{i+j \times nrClients} : j = 0, 1, ..., \frac{p}{nrClients} - 1\}$. Figure 6 shows an example of this division. In this example each client will receive four small sections of the image. The image sections a single client will receive are evenly distributed through out the image to be generated (e.g. Figure 6 clearly shows that the first client will receive grid sections 1,5, 9, 13). In this case the computation of the regions with significant details is evenly distributed amongst processors.

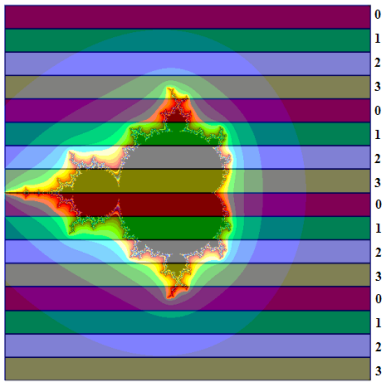


Figure 6: Cyclic Division of Image Area.

3.2.3 Dynamic Load Balancing

The Server maintains a Work Pool of jobs that can be sent out to clients (Figure 7). Initially every connected Client will receive a work unit to process, as soon as a Client returns a result another work unit is issued to the Client. This process is carried out until all the jobs in the Servers Work Pool have been completed. For this procedure the image area is typically divided into a grid structure. There is of course a slight increase in communication costs compared with that of the Uniform Block method, but the distribution of processing should be far more even across all connected Clients.

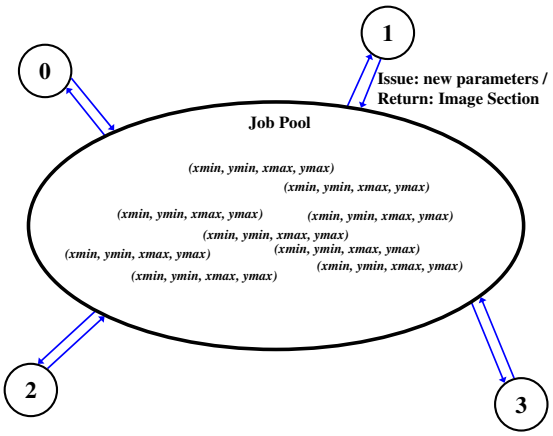


Figure 7: Dynamic Division of Image Area.

The division of the image into a grid (Figure 8) is carried out by the user choosing a particular granularity g which is the width of each grid block. Figure 8 is a 200 x 200 pixel image divided into 16 areas (Granularity of 50 pixels). The granularity by which the image is divided greatly effects the number of messages that are sent between Server and Clients (Figure 9). The area (in pixels) of each grid block is given by g^2 so that the number of grid blocks is given by $\frac{w \times h}{g^2}$. The total number of exchanges between Server and all the Clients has the form $2\left(\frac{w \times h}{g^2}\right)$ as a request must be sent to the client and a result returned. The total number of exchanges between the Server and any connected Client is $2\left(\frac{w \times h}{g^2 \times nrClients}\right)$. It is clear that the granularity has an important effect on both computation and communication. For computation the finer the granularity is the better the computation load balance becomes amongst processors. With a fine granularity the grid blocks have a small area so that the clients compute them in short time. In the case that one client becomes idle (work pool is empty) this will wait only little time for the other clients to finish the computation. Moreover, a fine granularity will evenly distribute the regions of high details amongst all the processors. However, fine granularity increases the amount of communication between the Server and the Clients. This would imply more overheads to start up the communications as well as more information that is sent between the Server and the Clients.

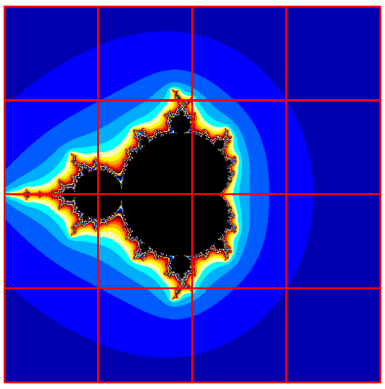


Figure 8: Dynamic Division of Image Area.

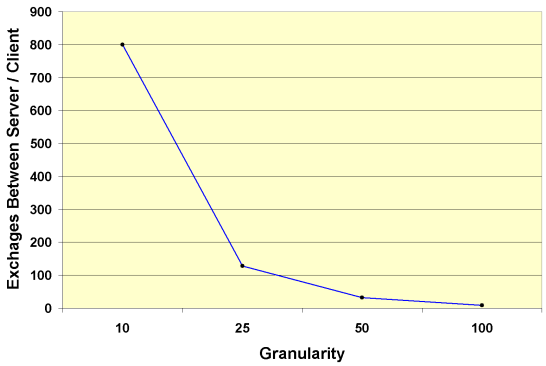


Figure 9: Number of Exchanges Required as Granularity Varies

3.3 Client / Server Operations

Overall the general methodology of the system is quite simple (Figure 10). The initial stages of the process are carried out on the Server. Firstly it is necessary to acquire the Input Settings for the Fractal Image, a Graphical User Interface (GUI) (Figure 4) is provided for this. When the user issues a request to generate a Fractal Image the parameters are gathered from the Fractal Image Settings GUI. The next stage is to calculate the parameters necessary for each client (this will depend on the number of clients currently connected). This yields a unique set of parameters for each client. Several other parameters are also passed which are the same for all clients (for example: formula type, number of iterations).

Once all the parameters have been finalised the operation of sending the Image Parameters to each connected client can commence. The parameter data is passed in the form of a string. A typical example of this string has the format of "width, height, xmin, ymin, xmax, ymax, iterations, equation type, cPower, zPower, invert, image segment number ". An example of the output string would be: "50, 200, -1.0, -2.0, 0.0, 2.0, 500, 0, 1, 2, 0, 1". The previous string would generate an image 50 x 200 pixels in size. The complex plane coordinates are "-1.0, -2.0, 0.0, 2.0". The client would carry out 500 iterations at each point. The generated Image would be the standard non inverted mandelbrot set $Z^2 + C$. The final parameter "image segment number" allows for the correct ordering of segments on the Server side.

The Client has in the meantime has been waiting for requests from the Server. Once a Client receives a request it must first parse the data to extract all of the required parameters necessary to generate the image. The next and most important stage is the actual generation of the fractal image. Each client will generate a small section of the image. The image section is then sent to the Server in the form of a sequence of integers using a DataOutputStream Object.

On the server side once it has issued its requests to all clients, it simply waits for incoming results. When a message is received from a client, the server examines the "image segment number" so the image will be placed in the correct order. Next it finds the length on the remaining incoming data, and initialises an array to be able to read all of the integer values representing the actual image. Once all the integer values have been read an Image object is created and positioned into is proper location based on the "image segment number". The process of waiting for client responses continues until all Image Sections are Retrieved. When the final image section is retrieved the Server displays the completed image on screen to the user.

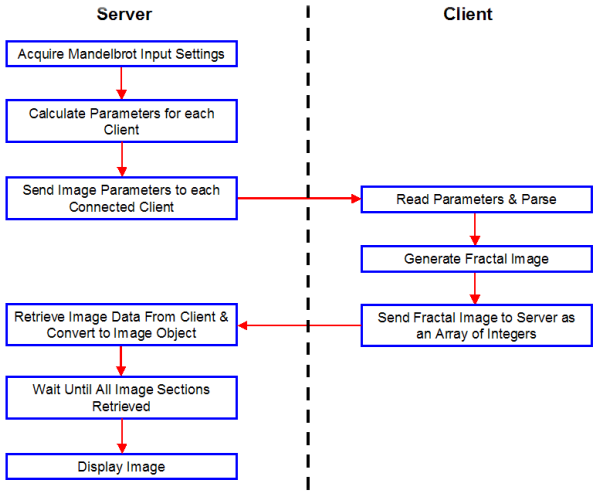


Figure 10: Client / Server Operations.

3.4 Bluetooth Networking

Typically the first step in a Networked Bluetooth application is to discover other Bluetooth Capable devices within the catchment area (10 meters for a class 3 Bluetooth Device, 100 meters for a class 1 device). For a Bluetooth device to advertise itself as being available it must be in "discoverable mode". There are two differing forms of this mode: General Unlimited Inquiry Access Code (GIAC) and Limited Dedicated Inquiry Access Code (LIAC). If the device is to be generally discoverable then it should be set to GIAC mode else it may be set as discoverable in a "limited" manner by using the LIAC mode. In this application the Client Devices are set to be generally discoverable, and as such when the Server Device is started it's first task is to discover all local devices available within it's catchment area.

In many Client to Server Applications the client requests services from the Server. In this case the Server usually starts running first and begins a cycle of waiting to accept new client requests. This application however the operation of the Client / Server system is slightly different. As with most Bluetooth applications the Master adds each client into the piconnet. The Master however issues requests to all client on the piconet (for example generate a fractal image) the client devices then carryout the actual processing work and return results to the Master (Server). This is akin to how SETI@Home [SETIatHome] works where the clients carry out the processing and return the results to the Server Application. An invaluable aid in the development of this system came in the form of a simple application from Nokia that demonstrated the use of a Point-multi-Point configuration [Nokia 2004]. Sony-Ericsson [Sony-Ericsson 2004] also have some very useful Developers Training Material dealing with Bluetooth Applications programming.

3.5 Execution Results

The experimental tests of this distributed fractal generating were carried out using 4 of Nokia 6630 phones. The scheduling partitions used in the experiment were *Uniform Block*, *Cyclic*, *Dynamic Scheduling with* $g = 50$ *and Dynamic Scheduling with* $g = 25$. The Mandelbrot fractal was generated into a 200 pixel square image to cover the plane region $[-2,2] \times [-2,2]$ with 500, 750 and 1000 iterations. Recall that the generation of the fractal on a single Nokia 6630 devices at 500 iterations requires 55,657ms to process, a figure far in excess of the execution times that will be presented below. Even when the Sony-Ericsson Wireless Toolkit (WTK) 2.2. emulator (Sony-Ericsson P900 Mobile Phone) was used the overall results showed a significant improvement in rendering time (see Tables 3 and 1).

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	13,675	4,453	11,456	7,455	2,235
750	14,047	4,609	11,838	7,422	2,557
1,000	14,907	4,437	12,101	7,469	2,610

Table 3: Image Generation Times for the Mandelbrot Set using the Sony-Ericsson WTK Emulator

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	25,878	8,246	20,457	13,379	4,217
750	26,051	8,230	21,064	13,277	4,542
1,000	26,442	8,311	21,418	12,945	4,547

Table 4: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Uniform Block Partition)

Table 4 presents the execution times for the *Uniform Block* scheduling. One can see that the second and third clients will receive the regions with more details so that they generate the highest computation times. While the forth client receives the region with lesser details so that it has the smallest execution time. As consequence those execution times show a huge load imbalance of the computation.

The use of the *Cyclic* scheduling corrects this load imbalance. The primary advantage with this scheme is that all areas of the image are distributed uniformly onto all the connected clients. The result of this is that areas where there is high computation cost are carried out by all processors. This means that the Server no longer has to wait for one or two nodes to finish the computation well after all other nodes have completed their assigned tasks. Table 5 shows the processing times for the cyclic scheduling scheme are very well balanced.

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	18,734	14,038	14,740	13.734	14,251
750	19,042	14,734	15,183	14.361	14,793
1000	19,623	14,829	15,053	14.853	15,391

Table 5: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Cyclic Scheduling Algorithm)

The *Dynamic Load Balancing* method was tested for two granularities. When $g = 50$ the fractal was splitted up into 16 grid blocks so that each client gets 4 blocks to compute. The computation times (see Table 6) show a very good load balance but they are bigger than the those of the *Cyclic* scheduling. In this case each client has at least 3 idle periods waiting for the communication with the Server to complete. Similarly, for granularity $g = 25$ each client has 16 grid blocks to compute so that it has 15 idle periods of waiting for communication. This however increases the computation times of each processor with around 8 seconds as Table Table 7 presents.

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	23,475	20.243,	20,048	21.482	21,729
750	23,739	21,473	21,762	21,906	21,845
1000	23,957	21,883	21,834	22.045	21,983

Table 6: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Dynamic Scheduling Algorithm with granularity $g = 50$ pixels)

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	30,754	28.393,	28,851	28.703	28,929
750	31083	30,710	29,675	29,496	30,025
1000	31,672	30,031	29,871	29.582	29,832

Table 7: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Dynamic Scheduling Algorithm with granularity $g = 25$ pixels)

Certainly, these experimental tests showed that the distributed computation over a piconet with Bluetooth reduces the generating time. Table 8 shows the reduction of the execution times for 500 iterations when the number of clients is $nrClients = 1, 2, 3, 4$.

	1 Client	2 Clients	3 Clients	4 Clients
Block	55,657	43,864	32,704	25,878
Cyclic	55,493	36,586	23,193	18,734
Dynamic	55,723	38,741	29,704	23,475

Table 8: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones for $nrClients = 1, 2, 3, 4$.

3.6 Further Work

Many alternative possibilities exist in terms of load balancing, examination and implementation of other scheduling schemes could provide noticeable improvement in the overall processing time. The implementation of alternate schemes would result in higher communication costs.

Another possibility is to make use of Scatternets. From the results so far it is clear that sections of the fractal image that contain a high level of detail require much more processing than areas of far less detail. So for the areas that require significant processing those sections could be distributed to a client that also acts as a server for another piconet.

4 Conclusion

A method of carrying out distributed fractal image generation across a piconet has been developed. This method of distributed fractal image generation has shown that it is capable of increasing the rate at which fractal images can be rendered on a mobile device.

Several methods of load balancing were implemented: Uniform Block, Cyclic Load Balancing and Dynamic Load Balancing. The Cyclic and Dynamic forms produce a far more even distribution of the work among the connected clients. It is clear from Tables 4, 5, 6 that the processing times reduce dramatically compared with the processing times with a single mobile device, for example 55,657 with a single Nokia 6630 (500 Iterations).

The method outlined for distributing the processing of a processor intensive task could be used in many other areas besides fractal image generation. The continually increasing number of mobile devices may prove to be a useful processing resource in the future. In time mobile devices may contribute to projects such as Seti@Home, DNA Analysis, Prime Number Search's to name but a few.

References

BOOK, M. Parallel fractal image generation. `http://www.matthiasbook.de/papers/parallelfractals/`.

DOOLAN, D. C., AND TABIRCA, S. 2005. Interactive teaching tool to visualize fractals on mobile devices. In *Proceedings of Eurographics Ireland Chapter Workshop*, Eurographics Ireland Chapter, 7–12.

KLINGSHEIM, A. N. 2004. *J2ME Bluetooth Programming*. Master's thesis, University of Bergen.

LONG, B. 2004. *A Study of Java Games in Bluetooth Wireless Netowrks*. Master's thesis, University College Cork.

LU, N. 1997. *Fractal Imaging*. Academic Press, San Diego, London, Boston.

LUCAS, G., AND MARQUAND, R., 1983. Star wars, return of the jedi. DVD Release Sept 2004.

MEYER, N., 1982. Star trek: The wrath of khan. DVD Release May 2003.

MPI. Message passing interface. `http://www-unix.mcs.anl.gov/mpi/`.

MURTY, R. 2004. Juliet: A distributed fault tolerant load balancer for .net web services. In *Proceedings of International Conference on Web Services (ICWS'04)*, IEEE.

NOKIA. Nokia platforms. `http://www.forum.nokia.com/main/0,6566,010,00.html`.

NOKIA, 2004. Introduction to developing networked midlets using bluetooth. `http://www.forum.nokia.com/info/sw.nokia.com/id/c0d95e6e-ccb7-4793-b3fc-2e88c9871bf5/Introduction_To_Developing_Networked_MIDlets_Using_Bluetooth_v1_0.zip.html`.

O'MAHONY, C. 2004. *Distributed Multimedia Processing*. Master's thesis, University College Cork.

SETIATHOME. The search for extra terrestial intelligence at home. `http://setiathome.ssl.berkeley.edu/`.

SONY-ERICSSON, 2004. Developing applications with the java api's for bluetooth (jsr-82). `http://developer.sonyericsson.com/getDocument.do?docId=65246`.

SONY-ERICSSON, 2005. Java platforms for sony-ericsson. `http://www.sonyericsson.com/developerimages/javaplatformversionsandscreensizesjune2005.xls`, June.

SUN-MICROSYSTEMS. Sun java studio mobility 6 2004q3. `http://www.sun.com/software/products/jsmobility/`.

SUN-MICROSYSTEMS. Windows wireless toolkit. `http://java.sun.com/products/sjwtoolkit/download-2_2.html`.

Developing Mobile 3D Games Using MIDP 2.0 Game API and JSR 184 Mobile 3D Graphics (M3G) API

Yu Han
School of Computer Engineering
Nanyang Technological University

#45-6-861 Hall of Residence 9
24 Nanyang Avenue, NTU
Singapore 639811
Telephone number 65-93363651
yuha0002@ntu.edu.sg

Abstract
This paper discusses in details how to apply the JSR 184 M3G API and the MIDP 2.0 Game API in the development of mobile 3D immersive games based on two games developed by the author. The use of third party content creation software, the problems of importing .m3g format resource files and the solutions proposed by the author will also be addressed.

As the processing power of mobile phones is still not up to the requirement of 3D immersive games, it is essential that the games are properly optimized to offer a satisfying gaming experience with reasonably fast frame rate. The optimization techniques which were employed to solve specific problems arising during the development of the sample games will be illustrated in detail.

Keywords
MIDP 2.0 Game API, Mobile Game Programming, Mobile 3D gaming, JSR 184 M3G API.

1 Introduction

The Java™ Mobile 3D Graphics (M3G) API has the power to turn a good game into an outstanding mobile gaming experience and profoundly change the way of presenting information on mobile devices. As the standard implementation of JSR 184 matures and the Java 3D enabled devices become widely available, mobile 3D games are starting become common commercialized applications instead of just being research topics in the laboratories.

Although proprietary 3D graphics software does exist in the current market, the trend among the mobile industry is to implement and support a common 3D graphics API that will facilitate portability of the applications and ease the process of development. For Java, the JSR 184 M3G API is considered the most suitable candidate. This paper discusses in details how this API combined with MIDP 2.0 Game API can be used not only to develop 3D immersive mobile games but also to improve the user interface of the game, while in some cases increasing the memory footprint of the application.

Moreover, the limitations of memory and processing power still exist for mobile devices and must be properly addressed in order for computationally expensive applications such as 3D games to run. This paper will also discuss several optimization techniques which can be employed to improve the performance of the games.

2 Exposition

2.1 Building Lightweight GUIs with MIDP 2.0 Game API

Previously, the GUI of the mobile games for the control and game settings were written simply based on J2ME built-in container classes such as Form or List. However, this has become increasingly inadequate given the high demand from consumers for more visually appealing GUIs. Using proprietary software to create GUIs requires the developer to learn to program in their specific ways and may sometimes involve licensing problems.

One way to solve this problem is to use the J2ME MIDP 2.0 Game API to write GUIs tailored to the specific game. The Game API offers excellent support in manipulating graphics and animations. If implemented as a Runnable thread, it enables the use of polling technique to simulate key events.

Even though the APIs allow porting applications to different devices, even across vendors, the size of the display varies from device to device. Therefore the UI elements should not be positioned using absolute coordinates, but in relative coordinates based on the screen size and the size of the content. For example,

X =((screen width) – (string width))/2

Figure 1: User menu implemented with Game API

The required information can be extracted from Font and GameCanvas classes. With each up or down key press, the indicator is moved by a predefined amount along the vertical axis to correspond to different choices. Internally, an index is updated to log the information of which option is currently highlighted. With the press of the select key, the current thread is terminated and the respective function handling the event is called.

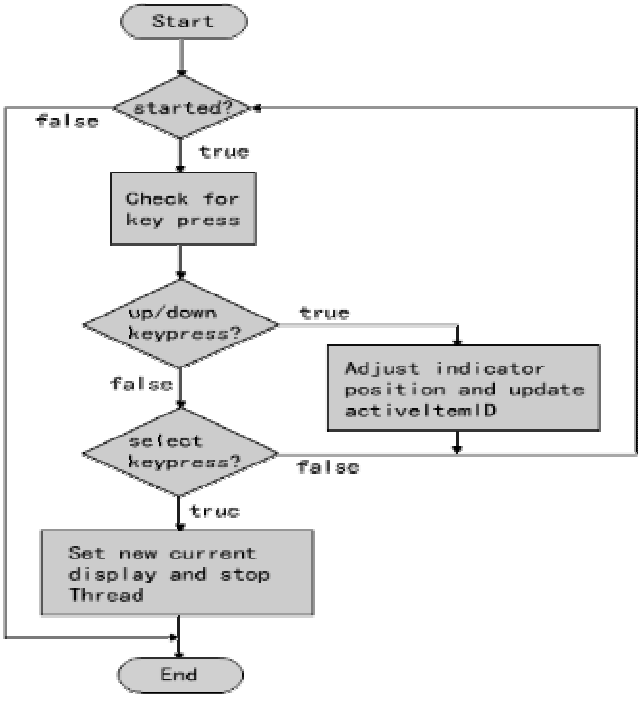


Figure 2: Flowchart of simulating key events using polling

When large numbers of resource files are being loaded, the game may appear not responsive for a long time. Users may suspect their mobile devices are crashed by the program and undesirable actions such as shutting down the device may be taken. To add in more interactivity, a progress bar can be implemented using J2ME MIDP 2.0 Game API to show the progress of loading the game. In the loading function, set different points to update a global integer value which represents the percentage of the game loaded. Then in the second thread, keep reading this value and redraw the progress bar. In this case, a separate thread is mandatory for the loading and drawing of the progress bar to be executed concurrently.

Figure 3: Progress bar for loading the game

2.2 Loading M3G Files

M3G files are a specially defined group of files which caters specifically to the JSR 184 M3G API. Objects or even entire scenes created with content creation tools such as 3D Studio Max or Maya can be exported as .m3g files and used by mobile games. This enables the games to have fairly complex objects or scenes which, in turn, enhance the gaming experience of the players.

However, loading a .m3g file may not be so straight forward as it seems to be. There is a discrepancy of implementation between the scene graph hierarchy of the .m3g file format and the JSR 184 M3G API: in the API, the root of the scene graph is an object of class World and all other 3D objects are children of this World object. But in the .m3g exporter, the root of the scene is an instance of the class AnimationController. Because of this difference, the developer must do a manipulation in the program to extract the World object out.

As each .m3g may have different indices for different objects, there is no way to be certain which index corresponds to the World object. In order to extract the desired object, the World object must be found first. Therefore, we use the following piece of code to locate the World object:

```
Object3D[] roots = Loader.load( filename );
World world;
for( int i = 0; i < roots.length; i++ ){
        if( roots[i] instanceof World ){
                world = (World)roots[i];
                break;
        }
}
```

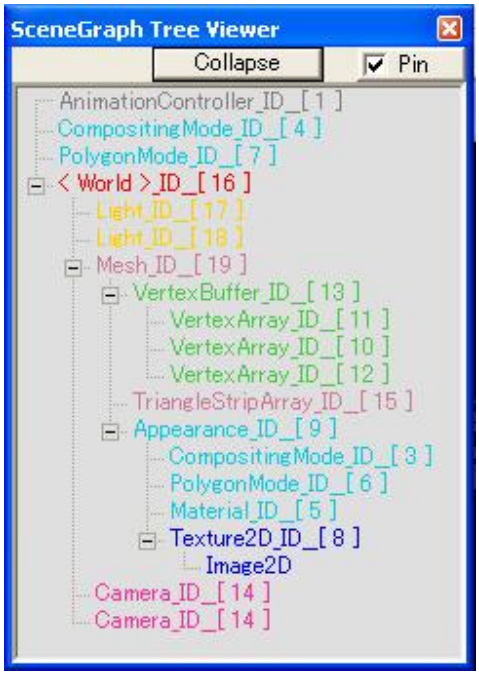


Figure 4: Scene graph of a .m3g file

Loop through the objects and find the only one which is of World type. As only one World object is allowed in each .m3g file, we can be certain that the World object is found when we first encounter it and exit the loop.

Another problem associated with .m3g file arises from the fact that each 3D object can only be child of one World. When multiple instances of the same object need to be present in the same World, the resource file needs to be reloaded and the object re-extracted which takes considerable amount of time. This problem will be addressed in detail in the later optimization section.

2.3 Perspective Distortion

When using graphics primitives such as triangleStrip to construct planar surface, perspective distortion may occur during texture mapping

Each planar face of the wall in Figure 5 consists of 2 triangles. When drawing the pixels of the texture, the positions of the ones near the upper edge are calculated with respect to the upper edge and the positions of the ones near to the lower edge are calculated with respect to the lower edge. Since these two edges appear to be not parallel from this perspective view, distortion occurs. The line separating the two triangles can be seen clearly in the figure below.

Figure 5: Perspective distortion

To alleviate this problem, more triangles can be used to represent one planar surface. As more and more triangles are added in, the effect of perspective distortion appears to be less and less noticeable. However, this method not only consumes more memory to store the extra triangles, it is also tedious for programmers to implement. Instead, the M3G API provides a function called setPerspectiveCorrectionEnable(boolean enable) in the PolygonMode class which eliminates the perspective distortion. However, the perspective correction flag is only a hint, so some implementations may not respect it.

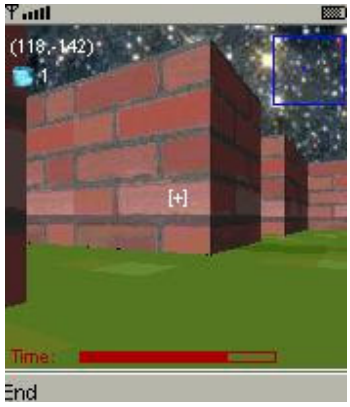


Figure 6: Perspective correction

2.4 Collision Detection

The JSR 184 M3G API provides a pick() function under the Group class for collision detection by ray intersection. An imaginary ray is cast from the center of the camera to infinity and the first mesh surface intersecting it at a predefined distance is considered causing a collision. This method is considered sufficient when navigating in a complex scene setting and there are too many objects to test for potential collisions by implementing bounding box or bounding sphere. However, the ray cast is only along the same direction as the camera. Therefore, when trying to detect collision during backward motions, the camera has to be temporarily reversed, test for collision, then reversed back.

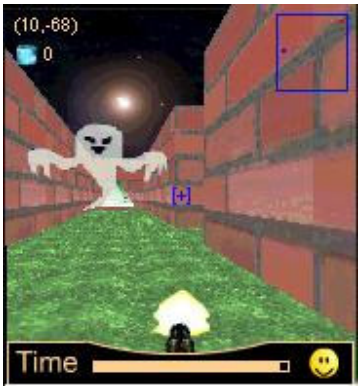


Figure 7: Collision detection by ray casting for firing of ammunitions

However, the pick() function may be implemented differently by some mobile phone manufacturers (I have tried using the pick() function with SonyEricsson phones and the collision detection did not work). Moreover, it is not suitable to use when the ammunitions are to be seen flying away (e.g. firing a missile). Therefore, in this case, bounding box or bounding sphere should be used.

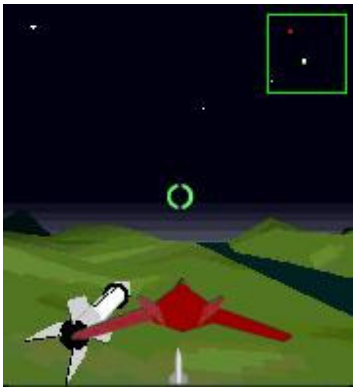


Figure 8: Bounding box collision detection

For mobile devices with limited processing power, bounding box is a better choice since it requires less calculation to detect a collision:

$$(X1-X2)^2 + (Z1-Z2)^2 + (Y1-Y2)^2 < distance^2$$

In the case when all the objects appear on roughly the same level, a bounding cylinder can further simplify the calculation and thereby improve the performance of the game:

$$(X1-X2)^2 + (Z1-Z2)^2 < distance^2$$

In this case, a cylinder along y-axis with infinite height is used for collision detection.

2.5 Optimization

Due to the limitation of memory and processing power of mobile devices, the games should always be optimized in order to make efficient use of the resource and achieve better performance. This is especially the case for 3D games which generally require more expensive computations than 2D games when rendering the screen.

First of all, not all the components have to be 3D. The graphics shown on the screen are basically a way of presenting the information and logic of the game, so whenever possible, developers should resort to 2D graphics which generally render faster. For example, in Figure 8, there is a jet fighter image attached to the camera. It banks to the left or right when the player presses the left or right key respectively as shown in Figure 10. To use a 3D model to accomplish this requires a lot of computations when rotating the plane and rotating the camera about the plane. Instead, we use a 2D image with the various positions of the plane pre-captured and place the corresponding image on the screen when necessary.

Figure 9: 2D images of the plane

Figure 10: Plane banking to the right

Thread objects should not be used unless absolutely necessary. By creating too many processes running concurrently, the overhead of context switching can make a 3D mobile game run unbearably slowly. Therefore, sometimes it is a good idea to manage all the moving objects in several Vector objects and move all of them in the same Thread running the game loop.

As mentioned in earlier section, the M3G API disallows one Object3D instance to be child of multiple Worlds. By default, there is a World object in every .m3g resource file and even if one only wants to export a 3D object, it is automatically attached to this default World.

The most intuitive way to solve this problem is to load the resource file each time one new instance of the object is needed, extract the object, remove it from the old World and attach it to the World in which the game is set. However, loading external resource files takes a long time and this can cause a significant delay during the game play, resulting in disruptive and unpleasant gaming experiences.

To eliminate the delay, a concept similar to double buffering has been employed in one of my games. For each 3D object for which multiple instances might be needed, one copy of it is stored in the memory throughout the playing time of the game. Each time a new instance is required, the original copy is duplicated and the new copy is attached to the World. To do the duplication, the duplicate() function of Object3D class is used. It not only creates an exact replica of the original object but also set the parent to null if the object is a Node. Then this new copy can be a child of the game World without causing any error.

Although this method consumes more memory by saving one copy of each object regardless of whether it is being used, it eliminates the need to reload external resource files and thereby remove the long pauses during reloading. Overall, the performance of the game is improved.

Alternatively, the components of the object like IndexBuffers and VertexBuffers of a mesh can be replicated and the mesh reconstructed later. However, to locate these pieces of information in the .m3g exporter file requires the knowledge of the indices of them. And more often then not, the sheer amount of IndexBuffers and VertexBuffers involved in complex models would make implementing this method a daunting task.

3. Observation

Although there have been significant improvements in the processing power, memory capacity and floating point support in the recently launched mobile phone models, the 3D graphics performance does not offer a very satisfying game play experience especially in the case of a first-person-view game.

However, with the proper application of the optimization techniques mentioned above, an approximately 30% increase in frame rate has been achieved.

Phone Model	**Frame rate before optimization /FPS**	**Frame rate after optimization /FPS**
SonyEricsson K300	6.5	11
SonyEricsson K500	7	10.7
SonyEricsson K700	8.3	12.5
SonyEricsson F500i	7.6	11.5

Table 1: Performance of the same game before and after optimization on various phone models

The values shown above are average values for one complete game session on actual devices. With further refinement in optimization techniques and continued improvement in mobile phone hardware, the 3D immersive games developed with JSR 184 M3G API will surely achieve a satisfying frame rate of 20~25 FPS and offer a new outlook to the mobile gaming industry.

References:

SonyEricsson general article, *Java 3D - a new opportunity in mobile gaming*, March 9, 2005

Qusay H. Mahmoud, *Getting Started With the Mobile 3D Graphics API for J2ME*, September 21, 2004

Sony Ericsson Developers Network, *Mobile 3D Graphics and Java Applications Development for Sony Ericsson Phones*, November 2004.

Sun Microsystem Inc, *JSR 184 Mobile 3D Graphics (M3G) API.* http://java.sun.com/j2me/docs/index.html

Tomi Aarnio, Kari Pulli, Nokia Research Centre,

Advanced Game Development with the Mobile 3D Graphics API

Alexei Sourin, *Computer Graphics – From a Small Formula to Virtual Worlds*, published in 2005 by Prentice Hall, ISDN 981-244-743-1

Experience of light, colour and space in Virtual Environments

A work in progress about how to make light and colour phenomena appear as in real rooms

Introduction

The actors in the design process have difficulties to visualize and predict how the not yet built environment is going to be experienced. Realistic virtual environments could make it easier for architects, users and clients to participate in the planning process of material choices, illumination and colouring.

Today, you cannot make realistic models with Virtual Reality (VR), since there is a lack of knowledge about how different parameters work regarding light and colour in rooms. If we can trust that different visual qualities are correctly reproduced in the virtual environments, VR can be used as a pedagogical tool in order to learn more about how light and colour work together.

This poster presents results of comparisons between real rooms and different VR-simulations and one tested solution to improve colour appearance.

Aim

THE AIM WITH THIS PROJECT is to develop the knowledge of how different aspects of the virtual environment affect the spatial experience. The focus of our research lies on simulating light and colour phenomena more correctly.

Figure 1. *View of the VR- model (incandescent light). The subjects found the colours in the room to agree fairly well with reality. However, the light areas appeared too greyish and the yellow ones appeared too alike in the VR-rooms. In reality they were perceived as many different colours. The numbers refer to the areas in Table 1.*

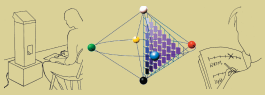
Methods

THE STARTING POINT FOR OUR STUDIES is the experience of a 25 m² multi-coloured real room, which has been compared with virtual models of the same room.The room was designed to get clear examples of how simultaneous contrast and reflections cause different appearances of the 6 different paints used. Three different light situations were used: tungsten, fluorescent 2700K and fluorescent 3000K. A digital model (3dsmax 6.0) of the room was made. This was exported to VR and showed stereographic and monographic on a desktop PC.

How light, colour and the spatial experience were experienced in the different light settings were investigated through video recorded interviews with 56 observers. People were asked to qualitatively describe the appearance of colour and light in the real room and in different simulations of the room. Their sense of presence and involvement were investigated through a separate questionnaire *.

Visual assesment techniques:

- Free description of the room as a whole
- Size estimation
- Semantic differential scaling with open scales (+ verbous motivation)
- Visual evaluation of light and colour
- Memory matching
- Colour matching



Physical measurements:

- The reflectance of each painted area was measured with a spectrophotometer. The L*a*b*-values were compared to the Lab-values of the digital colours.
- The spectral composition of the differently painted areas was measured with a spectroradiometer at different locations around the reference room and in the VR-room. We used the average values of L (Luminance), x and y (CI E 1931, 10 o) in the real room.
- The L, x and y-values were measured for the closest matching colour patch placed in the light box.

Results

IN THE REFERENCE ROOM, surfaces perpendicular to or opposite each other became more similar due to reflections. However, on each uniformly painted area, different colour variations were clearly visible. Moreover, between differently painted surfaces on the same level, effects of simultaneous contrast were evident. A brightness phenomenon appeared, i.e. a light surface in the darkest corner was perceived as whitest of all surfaces.

The interviews showed significant differences between the real and the virtual room. The VR-room had incorrect reflection effects between surfaces, too few colour variations and too achromatic shadows. The lightest areas were not simulated well; they became too grey. The contrast effects for the lightest surfaces were incorrectly reproduced. The one surface that was perceived as the whitest one in the real room was for example far too grey.

L*a*b*-values measured in reality could not be directly applied as digital Lab-values. Corresponding Lab-values on the display were too brownish for the light paints. The strong red and green needed to be adjusted in the opposite way. Otherwise the simulation became too brilliant and whitish.

The luminance was very different between the reference room and the VR-room. Nevertheless, the light level in the VR-room was assessed almost the same and the colours agreed fairly well. Spectral composition of the areas in the reference room and the matching colour patches in the light box were close. However, the corresponding areas in the VR-room had distinctly different L, x and y values.

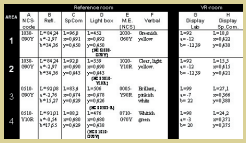


Table 1. *Reference room vs VR-room (flourescent light): examples of the collected data for four areas in the room, as well as the NCS-codes for the paints and results from magnitude estimation and verbal description.*

Defined problems

In the digital models there are:

- **Too few colour variations**
- **Too achromatic shadows**
- **Too small contrast effects**
- **The whitest areas are too greyish**
- **Incorrectly reproduced contrast effects**
- **Too simple chromatic information on light sources**

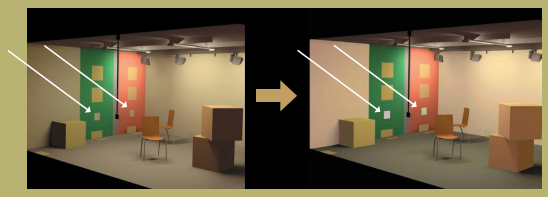


Figure 2A. *Appearance in 3dsmax 6.0.*

Figure 2B. *"Real" appearance (edited in Photoshop).*

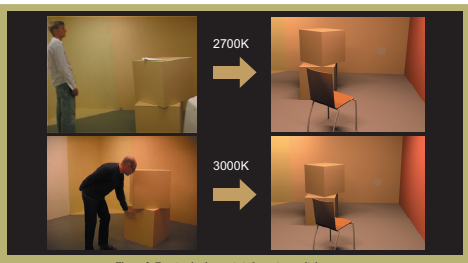


Figure 3. *Too simple chromatic information on light sources.*

Solution to one problem?

The focus of our research lies on simulating light and colour phenomena more correctly. A step in this process is to focus on problems concerning realistic reproduction of contrast effects in rooms.

In the digital models, some contrast phenomena did not appear.

In collaboration with the Dep of Information Technology at the University of Milan, we applied the Automatic Color Equalization (ACE)** algorithm to our models***.

ACE aims to:

- **Combine mechanisms of spatial interaction: lateral inhibition and local-global color induction**
- **Increase the dynamic range**

ACE applied on one image:

+ **Brighter squares**
+ **Brigher walls**
+ **Contrast phenomena appear on the small squares**

- **Too strong contrast effects**
- **Unrealistic contrast effects in the corners**

Figure 4. *ACE cannot stand alone as a method for filtering images of the whole room, because it treats the contrast effects between all surfaces as a whole, disregarding their location in space.*

ACE applied on rendered textures:

IN REALITY, there are contrast effects between coloured areas on flat surfaces, while angled surfaces affect each other by reflections. The problem with ACE is that it treats the contrast effects between all surfaces as a whole, disregarding their location in space. ACE is based on theories of 2D-colour phenomena. Our studies showed that it cannot be applied for rendering the complete 3D-model.

Therefore, in the process of making VR-models we used ACE for filtering rendered textures of multicoloured flat surfaces in a 3D model. Through this process, the light areas in the VR-model improved in colour appearance (see Fig 5A-C).

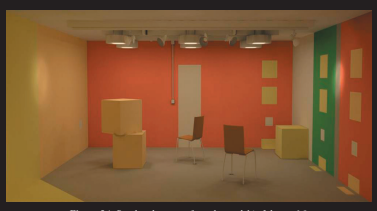


Figure 5A. *Rendered textures from the model in 3dsmax 6.0.*

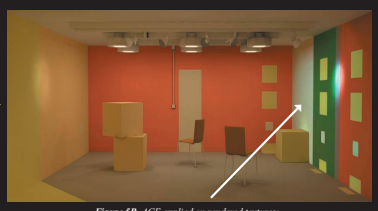


Figure 5B. *ACE applied on rendered textures:*
• Better contrast effects
• Increased brightness for the white areas

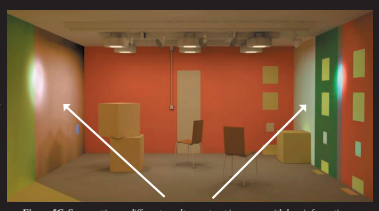


Figure 5C. *Same settings - different results; contrast increases with less information. This problem might be diminished by using anchor points with reference colours.*

Reflections do not exist in 2D, but are evident in 3D.
Therefore, theories on colour perception in 2D cannot automically be applied in 3D.

Conclusions

The study showed various problems related to the translation and comparison of reality to VR. The complexity of the task requires further systematic research in connection with elaborations in VE's. It is not enough to measure and mathematically model the physical conditions; we need to include psychological phenomena and the way our human vision works.

Problems to solve concern, for example, how to compare visual results between different medias, mixed adaptation and arbitrary parameter setting in the software.

ACE has showed to be able to reproduce contrast effects visible in reality, which are lost in the digital renderings. However, further adjustments of the ACE will be necessary. At this stage it exaggerates the contrast effects, which will have to be corrected in the parameter settings of the algorithm. We conclude that the ACE can be used as a starting point for a new algorithm.

Future work

A development of better algorithms is needed. We want an algorithm that can take into account the amount of information given in different parts of the image.

Our strategy is to list and specify spatial light- and colour phenomena, including comparing physical data to visual appearance of colours, in order to make them programmable. We plan to create an algorithm which is adjusted to the eye's perception of reality and the threedimensional vision.

Vision

- **Better balance between reflections and contrast effects**
- **Better represented spectral composition for the light sources**
- **Increased colour variations**

References

** M. Billger, I. Heldal, B. Stahre, K. Rennström, Perception of Colour and Space in Virtual Reality: a comparison between a real room and virtual reality models, Proceedings for IS&T/SPIE 16th annual conference on Electronic Imaging, San José USA, 18-22 Jan 2004, pp. 90-98*

** * A. Rizzi, C. Gatta, and D. Marini. A new algorithm for unsupervised global and local color correction. Pattern Recognition Letters, 24: 2003*

** * * Stahre B., Gatta C., Billger M.and Rizzi A., Towards perceptual colour for virtual environments, paper accepted for proceedings and presentation at AIC Granada 2005, 8-13 May*

CHALMERS

Beata Stahre & Monica Billger
Department of Architecture
CHALMERS UNIVERSITY OF TECHNOLOGY, SWEDEN

Contact: bea@chalmers.se and mpb@chalmers.se

LUND UNIVERSITY

ISBN 91-85457-86-8 (print)
http://www.ep.liu.se/ecp/016/

ISSN 1650-3686 (print)
ISSN 1650-3740 (online)