ISOLATING SELF-MOTION MANIFOLDS ON A PLAYSTATION™

J.M. Porta, L. Ros, F. Thomas
Institut de Robòtica i Informàtica Industrial (CSIC-UPC)
Llorens i Artigas 4-6, 08028-Barcelona, Spain
{jporta,llros,fthomas}@iri.upc.es

Abstract In this paper, we show how it is possible to use the specialized hardware of a Play Station 2 (PS2) to speed up the execution of the Cuik algorithm, an interval-based inverse kinematics algorithm. A PS2 includes a RISC CPU and two parallel vector units able to perform four floating point multiplications and four additions in a single cycle. A careful use of these two vector units to implement the basic operations of the Cuik algorithms allows increasing the speed of the algorithm in one order of magnitude. The final implementation in the PS2 is as fast as that in a Pentium 4 at 2.6GHz (that is at least three times more expensive than a plain PS2).

Keywords: Inverse kinematics, hardware implementation of algorithms, parallel algorithms, graphic hardware, Play Station 2.

1. Introduction

To speed up inverse kinematics computations we can either develop better algorithms and heuristics or provide better implementations of already existing ones. One possibility to speed up a given algorithm is to use special hardware to implement its basic operations. This is the line followed, for instance, by Lee and Chang, 1987, using the CORDIC architecture. In this work, the inverse kinematics of a particular mechanism is analytically solved and the resulting equations are efficiently implemented using the special operations provided by the CORDIC processors. A limitation of this work, however, is that it can only be applied to mechanisms whose inverse kinematics can be solved in closed form.

In Porta *et. al*, 2002, we introduced the Cuik algorithm, an interval-based algorithm able to deal with mechanism with an arbitrary number of kinematic loops. The basic operations of this algorithm are homogeneous matrix products. Nowadays, commercially available graphic cards include specialized hardware to perform products of homogeneous matrices. The problem is how to use this hardware to implement our algorithms. In general, the communication between the CPU and the graphic card is slow and unidirectional. Therefore, although we can per-

form some computations using this specialized hardware, it is hard to move back the results to the main CPU.

Recently, Sony has released a Linux kit for Play Station 2 (PS2) that makes its programming rather easy. The design of the PS2 differs in its principles from that of a PC (Diefendorff and Dubey, 1997). The PC was designed for *static applications*: large programs, such as word processors or spreadsheets, that deal with relatively static data (i.e., the document). PS2 was designed for *media applications*: small applications dealing with a constantly changing data stream. In a graphical context, a PC is designed work with a mainly static environment including a large number of geometric elements. This geometry is send once to the graphic card and, afterwards, many operations (viewpoint changes, etc) are applied to it. On the contrary, a PS2 is designed to repetitively apply the same type of simple operations to a continuously changing geometry. This has many implications in the hardware design. Thus, while the PC has large graphic memories (to store the static geometry) and a relatively slow communication bus between the CPU and the graphical card, the PS2 has a reduced graphic memory and a fast communication bus.

An interval-based algorithm (and the Cuik algorithm in particular) can be considered a *media application*: a relatively simple process (i.e., the box reduction) is repetitively applied to different inputs (the set of boxes and sub-boxes where we recursively look for solutions). Thus, the Cuik algorithm could substantially benefit from being implemented on a PS2.

In section 2, we briefly describe the Cuik algorithm. Next (section 3), we describe the hardware resources of the PS2 and how use them to implement the Cuik algorithm. In section 4, we show the improvements resulting of using the PS2 and we conclude in section 5 summarizing our work and pointing out some directions for further development.

2. The Cuik Algorithm

A kinematic loop yields an equation of the form

$$c(x_1, \ldots, x_n, \theta_1, \ldots, \theta_m) = I, \tag{1}$$

where c is a matrix function, $x_1, \ldots, x_n$ are variables involved in translations, $\theta_1, \ldots, \theta_m$ are rotational variables, and I is the identity matrix. Translations and rotations can be represented using homogeneous transforms. Thus, equation 1 can be seen as

$$K_1 H_1(x_1) \ldots K_n H_n(x_n) L_1 J_1(\theta_1) \ldots L_m J_m(\theta_m) = I, \tag{2}$$

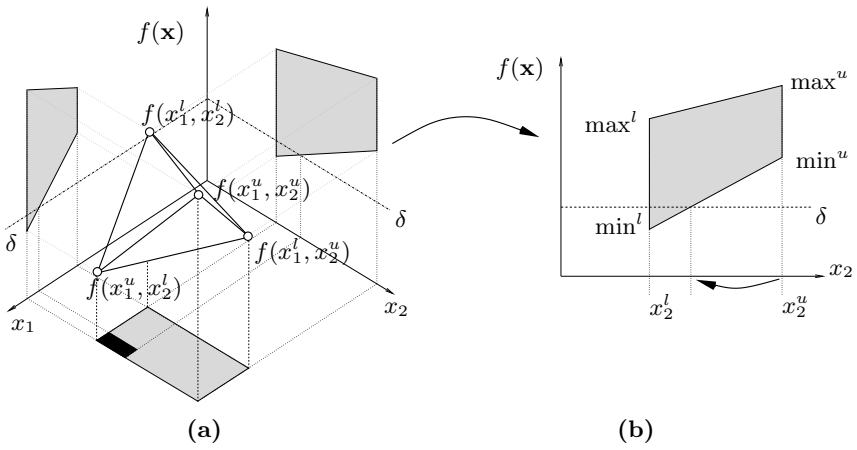

Figure 1. Segment-trapezoid clipping.

with K_i and L_i constant homogeneous transforms (possibly the identity), H_i a translation homogeneous matrix (along X, Y or Z) and J_i us a rotation matrix (around either X, Y or Z).

Each matrix equation leads to 12 scalar equations, one for each row i and column j in the transformation matrix:

$$f_{i,j}(x_1, \ldots, x_n, \theta_1, \ldots, \theta_m) = \delta_{i,j}, \tag{3}$$

where $\delta_{i,j}$ is 1 if i is equal to j, and 0 otherwise.

To avoid trigonometric expressions in the $f_{i,j}$ functions, we use the substitutions $y_k = \cos\theta_k$ and $z_k = \sin\theta_k$, and we add the circle constraint $y_k^2 + z_k^2 = 1$ to the system. The drawback of this variable change is that we have m additional variables and, thus, the search space is considerably enlarged. The advantage is that now all $f_{i,j}$ functions are multilinear, allowing us to use the following procedure to isolate the solutions of the resulting set of equations.

Assume we want to find all solutions of a multilinear equation $f(\mathbf{x}) = \delta$, for $\mathbf{x}$ in the box $\mathcal{B} = [x_1^l, x_1^u] \times \ldots \times [x_n^l, x_n^u] \in \Re^n$. According to Rikun, 1997, the graph of $(\mathbf{x}, f(\mathbf{x}))$ must lie within the convex hull of the 2^n points $\{(\mathbf{x}, f(\mathbf{x}))|\ \mathbf{x} \in \{x_1^l, x_1^u\} \times \ldots \times \{x_n^l, x_n^u\}\}$. To efficiently bound the intersection of this convex hull with the plane $f(\mathbf{x}) = \delta$ we project the hull onto each coordinate plane, as depicted in Fig. 1a for $n = 2$, and we intersect each of the resulting trapezoids with the $f(\mathbf{x}) = \delta$ line, as shown in Fig. 1b. Usually, these segment-trapezoid clippings reduce the ranges of some variables, and the Cartesian product of them all gives a box smaller than $\mathcal{B}$ still bounding the root locations (the black rectangle

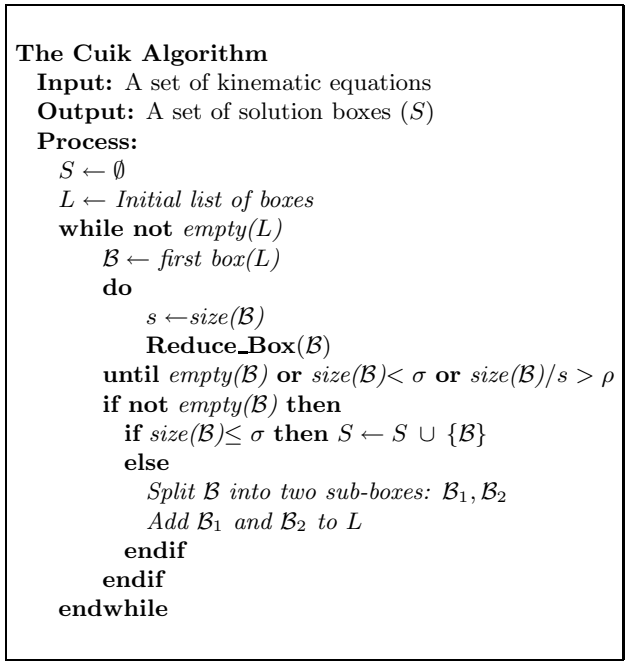

Figure 2. The Cuik algorithm.

in Fig. 1a). If the range of a variable y_k involved in a circle constraint $y_k^2 + z_k^2 = 1$ is reduced, then the range of z_k can also be reduced using a rectangle-circle clipping operation.

Fig. 2 shows the high level form of the *Cuik* algorithm. To determine the set of solution boxes, each of the initial boxes is reduced as much as possible by the successive application of the segment-trapezoid and rectangle-circle clippings (see Fig. 3). If one of the clippings yields no solutions, then the search can be stopped in the area delimited by the box in process. If a box becomes smaller than a given limit (σ) we consider it a solution. Finally, if a box can not be significantly reduced (its reduction ratio is below a user-provided parameter ρ), we split it and we process the two resulting sub-boxes in a recursive way.

3. Cuik on a PlayStation 2

The *Emotion Engine* (see Fig. 4) (the core controller of the PS2) is equipped with one RISC processor that works at 300MHz and two vector units (Sony, 2001). The RISC processor executes the operating system (Linux in our case). The vector units (VU0 and VU1) have two sub-processors. The first one, called Floating Point Unit (FPU), is

Reduce_Box($\mathcal{B}$)
Input: A box defined as a set of intervals:
$\mathcal{B} = \{[x_1^l, x_1^u], \ldots, [x_n^l, x_n^u]\}$
Output: The same box but eventually resized
Process:
for each kinematic loop c
$V \leftarrow \{v_0, \ldots, v_k\}$ *(Set of indices of variables involved in c)*
(Initialize the projection matrices)
for each $v \in V$
$\min_v^l \leftarrow +\infty$
$\min_v^u \leftarrow +\infty$
$\max_v^l \leftarrow -\infty$
$\max_v^u \leftarrow -\infty$
endfor
for each $\mathbf{x} \in \{x_{v_0}^l, x_{v_0}^u\} \times \ldots \times \{x_{v_k}^l, x_{v_k}^u\}$ *(For each corner of box $\mathcal{B}$)*
(Generate a sequence of matrices for this corner)
$S \leftarrow c(\mathbf{x})$
(Multiply the matrix sequence)
$s \leftarrow eval(S)$
(Project the matrix on the coordinate planes)
for each $v \in V$
if $\mathbf{x}[v] = x_v^l$ **then**
$\min_v^l \leftarrow \min(\min_v^l, s)$
$\max_v^l \leftarrow \max(\max_v^l, s)$
else
$\min_v^u \leftarrow \min(\min_v^u, s)$
$\max_v^u \leftarrow \max(\max_v^u, s)$
endif
endfor
endfor
(Perform the clippings)
for each $v \in V$
for each $i \in [1, 3]$ **and** $j \in [1, 4]$
if $i = j$ **then** $\delta = 1$ **else** $\delta = 0$
Trapezoid_Clipping$(x_v^l, x_v^u, \min_v^l(i,j), \max_v^l(i,j), \min_v^u(i,j), \max_v^u(i,j), \delta)$
if x_v is involved in a circle equation $x_v^2 + x_w^2 = 1$ **then**
Rectangle_Circle_Clipping$(x_v^l, x_v^u, x_w^l, x_w^u)$
endif
endfor
endfor
endfor

Figure 3. The **Reduce_Box** function.

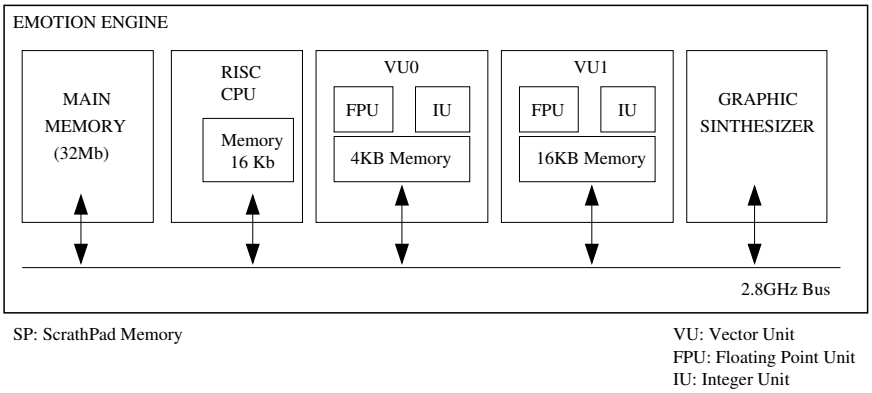

Figure 4. Sketch of the PS2 hardware.

specialized in the operation of vectors of four single-precession floating points elements. For instance, it can perform the element-wise product of two vector of four elements (i.e., four products) and accumulate the result to another vector (i.e., four additions) with a latency of 1 cycle. Using these operations, a full 4×4 matrix multiplication (as that performed in the *matrix sequence multiplication* step of the *reduce box* process) can be performed in just 16 cycles. Other basic operations the FPU can perform are the minimum and maximum of two vectors. We can use these operations to speed up the *matrix projection* step of the *reduce box* algorithm. The second sub-processor of the vector units, the Integer Unit (IU), can be used to feed the FPU with new data so that wait cycles are minimized.

The main memory of the Emotion Engine has 32 Mb. However, each processor has its own fast-access local memory. The bus communicating all the devices in the Emotion Engine operates at 2.8GHz, faster than the AGP graphic bus currently used in the PCs.

The special design of the PS2 enforces a particular programming philosophy. Small programs must be loaded in the VUs (with micro-instructions both to the FPU and to the IU). Then, the a continuous stream of data must be placed into the VU memories at the same speed as the VUs consumes the data.

To efficiently use the three processors of the PS2, we have to divide the Cuik algorithm in three sub-processes. Since, the VUs are specialized in matrix operations, we can use them to perform the matrix multiplications and the matrix projections included in the inner loop of the *reduce box* process (Fig. 3). The cost of both operations is $O(n)$, with n the

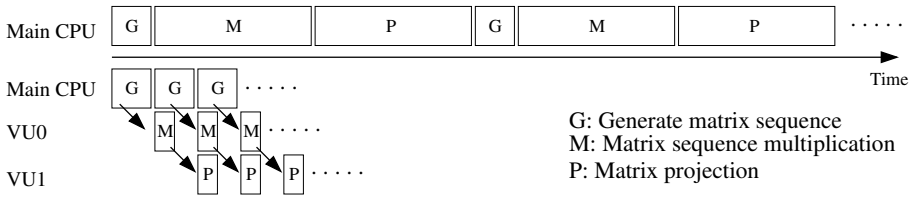

Figure 5. Time arrangement of the basic operation of the reduce-box process in a sequential processor (up) and in the PS2 (down).

number of variables involved in the loop. In this way, the load of the two VUs is balanced. The main RISC processor will take care of the rest of the algorithm: to generate the sequences of matrices to be multiplied (one sequence for each corner of the box in process, $\mathcal{B}$), to coordinate the execution of the vector units (sending data to them when appropriate), and to perform the clippings when the matrix multiplication and projections are done.

Fig. 5 shows a sketch with the arrangement of operations when using a sequential processors and when using the PS2. The use of three processors instead of just one, reduces the total cost of the algorithm at least by a factor of 3. Additionally, the time of the matrix multiplications and the matrix projection is considerably reduced due to the use of the special hardware of the VUs. The final result is a large increase of the speed in the algorithm.

4. Experiments and Results

Fig. 6a represents the Bennett linkage, a 4R spatial closed chain with 1-d.o.f. mobility. Fig. 6b shows the discretization of the one-dimensional solution space provided by the Cuik algorithm with $\sigma = 0.1$ and $\rho = 0.9$. This discretization includes about 300 boxes and it was achieved after applying the *box reduction* process 4150 times to 1575 boxes. Thus, the *box reduction* is applied, in average, 2.6 times to each processed box. The problem has 4 rotational variables but, after the sin/cos substitutions, we work with 8 variables. Consequently, each box has 256 corners and for each one of them we have to apply the *matrix sequence generation*, *matrix multiplication* and *matrix projection* steps.

If we apply the sequential version of the Cuik algorithm to this problem using the main processor of the PS2, the solution manifold of the Bennett linkage is isolated in about 265 seconds.

The parallelized version explained in the previous section, solves the problem in less than 22 seconds. So, using the special hardware of the

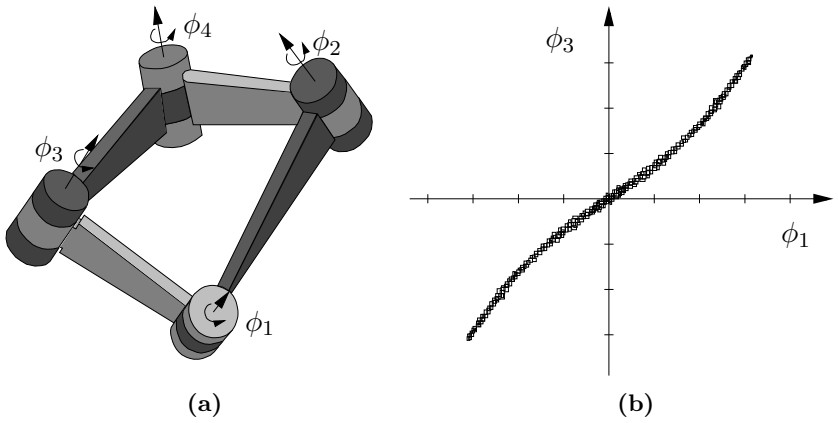

Figure 6. **(a)** The Bennett linkage. **(b)** The discretization of the one-dimensional space of solutions provided by the Cuik algorithm.

PS2 decreases the execution time to 10% of the original cost. The final execution time on the PS2 is roughly the same as that on a Pentium 4 at 2.6GHz, that is a much more expensive machine.

After the parallelization, the bottleneck in the PS2 implementation is the generation of the matrix sequences for each corner of all processed boxes. This takes about 97% of the final execution time. This process is that slow since it is executed by the RISC CPU of the PS2 that only operates at 300MHz. So, what in the sequential version of the algorithm was non-relevant (less than 10% of the execution time) becomes the most expensive step in the parallelized version. Therefore, any improvement in the matrix sequence generation procedure would result in a direct improvement of the global performance of the system.

The matrix multiplications and projections for the 4150 processed boxes are performed in less than 1 second. Thus, the PS2 achieves 1.8GFlops. This is less than the PS2 theoretical maximum (4.8GFlops) since not all instructions in the VUs perform floating point operations. (initializations, branches, etc are performed by the IU while the FPU is stopped).

5. Conclusions

Currently available graphic hardware offer many opportunities to speed up kinematic algorithms. In this paper, we describe the use of a PS2 to implement Cuik, a kinematic problems solver. Previously existing works on the use of hardware devices for inverse kinematic computations where

limited to particular mechanism while the Cuik algorithm can deal with (almost) general mechanisms.

The internal controller of the PS2 with three CPU's, two of them specialized in vector operations, happened to be perfectly tailored for the structure of the Cuik algorithm.

The use of the PS2 resulted in the parallelization of our algorithm at two different levels of granularity (Henrich *et al*, 1997). The first one is the in-box level: the different operations applied to each box can be implemented in a different processor of the PS2. The second level of parallelism we achieved is the operation-level: the basic floating-point vector operations are performed in parallel in the vector units of the PS2.

Additionally, the interval-based algorithms, such as Cuik, are inherently parallelizable at the box-level (different boxes can be processed by different computers). Thus, one possible extension of this work is the set up of a grid of PS2s to cooperatively solve large problems.

The version of the Cuik algorithm described in this paper is the most basic one. A future improvement is to incorporate to the PS2 version the heuristics and improvements we have already developed for other versions of Cuik. This would result in a even larger increase of the speed of the algorithm.

Another possibility we want to explore is the use of PC graphic cards to implement Cuik since they are more easily available in research labs than PS2.

Acknowledgments

This research has been partially supported by the Spanish CICYT under contract TIC2003-03396.

References

Diefendorff K. and Dubey P.K. (1997), *How Multimedia Workloads Will Change Processor Design*, IEEE Computer, Vol. 30, No. 9, pp. 43-45.

Henrich D., Karl J. and Wörn H. (1997) *A Review of Parallel Processing Approaches to Robot Kinematics and Jacobian* Technical Report 10/97, Computer Science Department, University of Karlsruhe, ISSN 1432-7864.

Lee C.S.G. and Chang P.R. (1987), *A Maximum Pipelined CORDIC Architecture for Inverse Kinematic Position Computation*. IEEE Journal of Robotics and Automation. Vol. 3, No. 5, pp. 445–548.

Porta J.M., Ros L., Thomas F. and Torras C. (2002), *Solving Multi-Loop Linkages by Iterating 2D Clippings*, Advances in Robot Kinematics. Kluwer Academic Publishers, pp.255–264.

Rikun, A.D. (1997), A Convex Envelope Formula for Multilinear Functions, *J. of Global Optimization*, Vol. 10, pp. 425-437.

Sony Computer Entertainment Inc. (2001) *The Emotion Engine User's Manual*.