7. International Immersive Projection Technologies Workshop
9. Eurographics Workshop on Virtual Environments (2003)
J. Deisinger, A. Kunz (Editors)

Haptic Rendering of Complex Force Fields

Aleš Křenek

Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
ljocha@fi.muni.cz

Abstract

With a particular focus in scientific applications, we propose a method of haptic rendering of virtual environments which require rather complex calculations to evaluate their behaviour. Therefore it is not possible to perform the calculations within a haptic loop. We specify a class of problems for which the method is applicable, describe the designed data structure, present involved algorithms, and prove their critical properties. The general section is followed by a case study of applying the method in a specific application in computational chemistry. The paper is concluded with first practical results of deployment of the method as well as preliminary quantitative assessment.

Cathegories and Subject Descriptions (according to ACM CSS): E.1 [Data Structures]: Graphs and networks, H.5.2 [User Interfaces]: Haptic I/O, J.2 [Physical Sciences and Engineering]: Chemistry

1. Motivation

For almost hundred years of their history movies utilize the human sense of sight inertion. Presenting a sequence of discrete frames at the rate of about 25 per second is sufficient to create an illusion of smooth motion. Increasing the refresh rate higher does not increase the perceived smoothness of motion.

On the contrary, the human sense of touch is at least two orders of magnitude more sensitive to the refresh rate. It has been found out[6, 7] that at least 1 kHz is required (which is a typical value for current devices), our experiments[1] show that in the case of modeling stiff smoothly curved surfaces the human is able to perceive increased quality to at least 5 kHz.

Consequently, the time available for a single iteration of the haptic loop, i. e. to evaluate the scene computational model based on the haptic device position and to compute the force (or vice versa) is limited to at most 1 millisecond. Even with current powerful CPUs the possible computation is limited to evaluation of simple model primitives, e. g. spring model of surface penetration[6].

In scientific applications, we aim at modeling complex phenomena for which the haptic user interaction may bring significant benefits over conventional approaches (plain visualization). However, computer models of such phenomena (e. g. finite element methods, quantum-chemical calculations etc.) tend to be computationally extensive. Assuming a 1 second calculation (which is still underestimation of many interesting problems) on a current CPU and the Moor's law, it would take about 15 years before CPUs were able to perform the calculation within the millisecond range. In the research presented in this paper we seek for overcoming this performance gap.

Assuming the typical case of three degrees of freedom of the haptic interaction a straightforward solution suggests to approximate the interaction force field with a discrete 3D grid. The model would be evaluated in advance in each point and the resulting forces stored. Once the grid were ready the haptic interaction could be run, interpolating among the precomputed points.

However, we claim that a 3D grid of precomputed points is not a sufficiently rich structure to cover interesting force fields. Let's consider trivial yet illustrative example, derived from the real application area described in Sect. 3: In a real-world environment (i. e. within a gravitational field) a ball is hung on a rope, vertically in its initial position. If the user approaches it with another ball horizontally from the left, finally arriving to the original position of the center of the ball, the ball is shifted to the right. On the other hand, if the user approaches the ball in the same way from the right, arriving to *exactly the same position*, the ball is shifted to the left (see Fig. 1). Thus we reached two distinct states of the system, resulting in different forces, but corresponding to the same input position. Despite very trivial, the system is rich enough so that its observed behaviour cannot be described completely by a simple force field, i. e. a function mapping the user's position to a force interaction. Consequently, we cannot capture it in a simple 3D grid where only a single state is stored in each point.

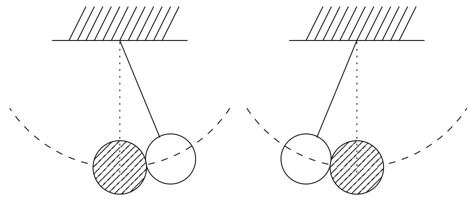


Figure 1: *When approached from opposite directions the hung ball (white) is shifted to different positions (system states), despite the final user position is the same in both cases.*

Our research is motivated mainly by potential applications found in computational chemistry. As virtually any chemical process is described with rules involving energy, various complex force fields are typical for expressing many chemical interactions. In Sect. 3 we show a concrete example — analysis of conformational behaviour of flexible molecules.

Our previous research[3] in this area proved that the general interaction paradigm we accepted was feasible. However, its deployment was limited to very simple systems due to the relative lack of computing power. Therefore we focused on developing a technique of separating expensive model-specific calculation from the interaction loop itself. The emerging results are discussed in the rest of this paper.

2. 3D Grid with Level Conversions

We present a method of haptic interaction with systems (or virtual environments) exhibiting the following features and constraints:

- The modeled system interacts with the user through an idealized single *haptic interaction point*[8] — HIP. The HIP is the only, two-way interface between the user and the environment, its position is taken as an input and the computed force feedback is applied at this point as well.
- The computation of the system behaviour can be virtually arbitrarily complex.
- All data required to describe the modeled system have to be known in advance. It is not possible to deal with environments influenced by e. g. real-time measurements. However, this constraint does not mean that the environment is static — we require only that the *rules* of dynamic behaviour has to be known.
- Any possible state of the modeled system can be described by a state vector, and the force interaction can be derived from such state vector.
- Behaviour of the system is described completely in terms of changes of the state w. r. t. the HIP position, and is independent on time. Consequently, we are currently not able to deal with time-related quantities (e. g. speed), include them in the system state, derive behaviour based on such quantities etc.

We developed a two phase approach to haptic rendering of such environments. The two (off-line and online) phases are as follows:

State space computation. The state space of the interaction with the given virtual environment is searched through systematically. The trace of the search is stored in a suitable data structure, an enriched 3D grid of sufficient resolution.
This computation is performed in advance, covers all application-specific expensive calculations, and may take rather long time.

State space traversal. Actual interaction with the given virtual environment is run using the precomputed data. Calculations required in this phase are limited mainly to linear interpolation and straightforward force computation eventually. Those can be done very fast, enabling real-time driving of the haptic device, as well as synchronous visualization of the environment.

In this section we focus on semi-formal description of behaviour of systems we are able to model with this approach. Then the intermediate data structure is described in detail, as well as algorithms involved in both the phases.

2.1. Modeled System States and Behaviour

Let's approach the problem more formally now. We consider a system for which an arbitrary state can be described completely by a *state vector* $\boldsymbol{s} \in \mathbb{R}^n$. Let's denote $A \subset \mathbb{R}^3$ the *area of interest*, i. e. the space of HIP positions where an interaction with the involved virtual environment is supposed to occur. Without any significant loss of generality we can assume that A is continuous and convex (it is a rectangular bounding box in most applications).

For each $\boldsymbol{x} \in A$ we denote $S(\boldsymbol{x})$ the set *feasible states*, i. e. the states of the system that are possible for a given HIP position $\boldsymbol{x}$. To be able to compute the state space we require that there is at least one *seed point* $\hat{\boldsymbol{x}} \in A$ such that at least one state $\hat{\boldsymbol{s}} \in S(\hat{\boldsymbol{x}})$ is known a priori.

Behaviour of the system is described by a *conversion function* $C\colon (\boldsymbol{s}_i, \boldsymbol{x}_i, \boldsymbol{d}) \mapsto \boldsymbol{s}_n$. Given an initial HIP position $\boldsymbol{x}_i$, initial state $\boldsymbol{s}_i \in S(\boldsymbol{x}_i)$ and a position shift vector $\boldsymbol{d}$, the conversion function computes the state of the system to which it arrives if the HIP moves directly to $\boldsymbol{x}_i + \boldsymbol{d}$. We require the following conditions on the function C to hold:

a) *Local evaluation*: There is a global constant $d_{\max} > 0$ such that for each $\boldsymbol{x} \in A, \boldsymbol{s} \in S(\boldsymbol{x})$ and $|\boldsymbol{d}| \leq d_{\max}$ if $\boldsymbol{x} + \boldsymbol{d} \in A$ then $C(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d})$ is defined.
This property guarantees that given a HIP position and system state, it is always possible to compute a state change corresponding to a small but non-zero position change. The computed 3D grid resolution is derived from the $d_{\max}$ value then (see Sect. 2.2).
b) *Completeness w. r. t. states*: For each $\boldsymbol{x} \in A, \boldsymbol{s} \in S(\boldsymbol{x}), |\boldsymbol{d}| \leq d_{\max}$ such that $\boldsymbol{x}+\boldsymbol{d} \in A$ the computed state $C(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d})$ falls into $S(\boldsymbol{x}+\boldsymbol{d})$. In other words, as long as moving within A it is not possible to reach an "unknown" state.
c) *Finite number of levels*: $S(\boldsymbol{x})$ is finite for each $\boldsymbol{x} \in A$. We are going to capture the entire state space by a precomputed data set. Hence we have to impose this restriction to prevent infinite computation.

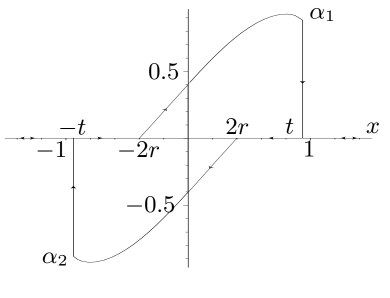


Figure 2: *State space of the example in Fig. 1.*

Let's consider the example from the previous section (Fig. 1). We assume the distance from the fixed point to the hung ball center is 1 all the time and denote $r \ll 1$ the radius of both the balls. We restrict the movement of both to one dimension only. Hence the state of the system is described uniquely by a single value α — the angle of the rope (being zero initially).

The state space of the system is shown in Fig. 2. As long as the user approaches from the left until $x = -2r$, the ball remains in the initial position $\alpha = 0$.

x	α	d	$C(\alpha, x, d)$
$(-\infty, -t]$	0	any	0
$(-t, -2r]$	0	$x + d \leq -2r$	0
		$x + d \geq -2r$	$\alpha_1(x + d)$
$[-2r, t)$	$\alpha_1(x)$	$x + d \leq -2r$	0
		$x + d \in [-2r, t)$	$\alpha_1(x + d)$
		$x + d \geq t$	0
$[2r, t)$	0	$x + d \geq 2r$	0
		$x + d \leq 2r$	$\alpha_2(x + d)$
$(-t, 2r]$	$\alpha_2(x)$	$x + d \leq -t$	0
		$x + d \in (-t, 2r]$	$\alpha_2(x + d)$
		$x + d \geq 2r$	0
$[t, \infty)$	0	any	0

Table 1: *Conversion function definition for the example in Fig. 1. The first three columns represent ranges or values of the function arguments, the last one is the resulting state.*

Then it follows the function α_1. From this point the distance between the balls centers remains $2r$, hence α_1 must be a solution of the equation

$$(1 - \cos\alpha)^2 + (x - \sin\alpha)^2 = 4r^2 .$$

The actual analytical solution is a rather complicated formula containing over 50 terms, and it is pointless to present it here. For the purpose of this example we computed it with Maple symbolic math software[5].

Eventually, the user reaches a tear-off point $t = 2\sqrt{r(r+1)}$ (this value is derived from the fact that the fixed point and both balls centers are collinear at this point), and the ball drops back to the initial position. The behaviour is symmetric when approaching from the right, following the function α_2 once the balls touch each other. Exact definition of the corresponding conversion function is shown in Table 1. In order to reduce the number of cases (rows) we assume $d_{\max} < t - 2r$. It can be seen that the above conditions a–c are met.

A conversion function conforming with the three conditions above describes completely the system behaviour and is sufficient to build a model of haptic interaction through the HIP. We do not impose constraints on time required to evaluate the conversion function. On the contrary, it should cover all the model-related time consuming calculations.

A force interaction between the user and the model in the HIP is described by a *force function* $F(\boldsymbol{x}, \boldsymbol{s})$ mapping the HIP position $\boldsymbol{x} \in A$ and a feasible model state $\boldsymbol{s} \in S(\boldsymbol{x})$ to a force that should be delivered to the user. It is convenient if the state vector is designed in such a way that F can be evaluated inside the haptic loop, i. e. in less than 1 ms, however, we do not require it strictly (see Sect. 2.2).

In addition, we assume that the vector $\boldsymbol{s}$ contains enough information to visualize the model in the given state, and it's also possible to do it sufficiently fast to get smooth visualization (i. e. at 25 Hz). If it is not possible for a given application, further visualization data has to be precomputed and stored as well.

2.2. Precomputed Data

The precomputed dataset serves as a discrete approximation of the described continuous conversion function. Its structure is based on a 3D grid of evenly distributed samples of the HIP position within the area of interest A. For the sake of simplicity, we assume further on that A is a rectangular box. Let's denote $\bar{d}$ the axial distance between adjacent grid points.

The grid is enriched with *levels* and conversions among its adjacent points. In each grid point $\boldsymbol{x}$ there are as many levels as is the number of feasible states $S(\boldsymbol{x})$. Then, each level l stores a record containing

- a state vector $\boldsymbol{s}_l$,
- $3 \times 3 \times 3$ cube D_l of *destination levels*,
- force vector $F(\boldsymbol{x}, \boldsymbol{s}_l)$ (optional),
- visualization data (optional).

The cube of destination levels encodes the conversion function. We ignore the centre of the cube and interpret its remaining 26 elements as vectors $\boldsymbol{d} \in \mathcal{D}$ where

$$\mathcal{D} = \Big\{(d_1, d_2, d_3) : d_1, d_2, d_3 \in \{-\bar{d}, 0, \bar{d}\} \text{ and } |(d_1, d_2, d_3)| \neq 0\Big\}.$$

For each such $\boldsymbol{d}$, the corresponding value in the D array indicates that the destination state $C(\boldsymbol{s}_l, \boldsymbol{x}, \boldsymbol{d})$ is stored at the grid point $\boldsymbol{x} + \boldsymbol{d}$ at level $D_l[\boldsymbol{d}]$.

The three principal requirements on the conversion function from Sect. 2.1 are reflected in this data in the following way:

a) Conversion function must be defined on a sufficiently large surrounding of each grid point to reach all the adjacent points, including volume diagonals. Hence $d_{\max} \geq \sqrt{3}\bar{d}$. Or, vice versa, if the $d_{\max}$ value is constrained by the conversion function properties, the grid resolution must be fine enough to accomplish this inequality.
b) For each grid point $\boldsymbol{x}$, level l at this point and a shift vector $\boldsymbol{d}$ such that $\boldsymbol{x} + \boldsymbol{d} \in A$, the destination level $D_l[\boldsymbol{d}]$ must exist at $\boldsymbol{x} + \boldsymbol{d}$ and the state stored there must be $C(\boldsymbol{s}_l, \boldsymbol{x}, \boldsymbol{d})$.
c) $S(\boldsymbol{x})$ elements are mapped 1:1 to the levels. Hence the requirement on finiteness is obvious.

The optional force and visualization data are included when the underlying model does not allow to compute them sufficiently fast (i. e. 1 ms in case of the force, and about 40 ms for visualization data) from the state vector.

2.3. State Space Computation

The algorithm computing the described enriched grid works with a queue Q of quadruplets $(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d}, l)$ — requests to store a data record consisting of a state $\boldsymbol{s}$ at the HIP position $\boldsymbol{x}$, having been found by conversion function evaluation from position $\boldsymbol{x} - \boldsymbol{d}$ and level l. Results are stored into the four dimensional array X. The code is shown in Algorithm 1.

Algorithm 1 Compute the enriched grid

```
 1: for all seed points $\hat{\boldsymbol{x}}$ do
 2:     $l := 0$
 3:     for all known $\hat{\boldsymbol{s}} \in S(\hat{\boldsymbol{x}})$ do
 4:         $X[\hat{\boldsymbol{x}}, l].\boldsymbol{s} := \hat{\boldsymbol{s}}$
 5:         expand_state($\hat{\boldsymbol{x}}, \hat{\boldsymbol{s}}, l$)
 6:         $l := l + 1$
 7:     end for
 8: end for
 9: while $Q$ is not empty do
10:     $(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d}, l') \leftarrow Q$
11:     $l := 0$
12:     while $X[\boldsymbol{x}, l]$ exists and $X[\boldsymbol{x}, l].\boldsymbol{s} \neq \boldsymbol{s}$ do
13:         $l := l + 1$
14:     end while
15:     if $X[\boldsymbol{x}, l]$ does not exist then
16:         $X[\boldsymbol{x}, l].\boldsymbol{s} := \boldsymbol{s}$
17:         expand_state($\boldsymbol{x}, \boldsymbol{s}, l$)
18:         compute and store force and/or visualization
                data if necessary
19:     end if
20:     $X[\boldsymbol{x} - \boldsymbol{d}, l'].D[\boldsymbol{d}] := l$
21: end while

procedure expand_state($\boldsymbol{x}, \boldsymbol{s}, l$)
  for $\boldsymbol{d} \in \mathcal{D}$ do
    if $\boldsymbol{x} + \boldsymbol{d} \in A$ then
      $\boldsymbol{s}' := C(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d})$
      $Q \leftarrow (\boldsymbol{s}', \boldsymbol{x} + \boldsymbol{d}, \boldsymbol{d}, l)$
    end if
  end for
```

Initially, the queue is initialized with one or more seed points (see Sect. 2.1) for which some states are known. Usually those are the corners of the area of interest.

A core of the computation is the procedure `expand_state`. Given a HIP position $\boldsymbol{x}$, state $\boldsymbol{s} \in S(\boldsymbol{x})$ and a level l where $\boldsymbol{s}$ is stored the procedure loops over all 26 directions and if the particular HIP position shift does not reach beyond the area of interest, the conversion function is evaluated in these directions, and the reached states enqueued to be processed later.

In each iteration of the algorithm main loop (starting at line 9), a queue head $(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d}, l')$ is extracted. First, the grid point $\boldsymbol{x}$ is checked whether the state $\boldsymbol{s}$ has been already stored there. If not, a new level l at $\boldsymbol{x}$ is allocated and the state $\boldsymbol{s}$ stored. Then the procedure `expand_state` is called to compute the conversion function (starting from $\boldsymbol{s}, \boldsymbol{x}$) and enqueue the results.

In either case the original record in $\boldsymbol{x} - \boldsymbol{d}$ at level l' (from where the conversion function was evaluated and the state $\boldsymbol{s}$ reached in some iteration before) is updated so that the destination level for the shift $\boldsymbol{d}$ is set to the discovered level l.

The algorithm terminates when the queue is empty. Let's sketch a proof that it always happens. Given a position $\boldsymbol{x}$, state $\boldsymbol{s} \in S(\boldsymbol{x})$, and position shift $\boldsymbol{d} \in \mathcal{D}$ a request containing $\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d}$ is enqueued into Q exactly once, upon storing $\boldsymbol{s}$ into $X[\boldsymbol{x}]$ (line 16). On all further occurrences of $\boldsymbol{s}, \boldsymbol{x}$ in the queue, no requests are enqueued. The number of distinct $\boldsymbol{x}$'s (grid points) is finite, and according to the requirement c) in Sect. 2.1 all the sets $S(\boldsymbol{x})$ are finite as well. Hence the number of requests ever enqueued is finite and the algorithm terminates consequently. $\square$

Further on, we show that upon termination the output data X are complete — accomplish the condition b) in Sect. 2.2: Let's choose an arbitrary $X[\boldsymbol{x}, l]$. When this record was created (line 16), the procedure `expand_state` was also called. Therefore for each $\boldsymbol{d} \in \mathcal{D}$ such that $\boldsymbol{x} + \boldsymbol{d} \in A$ a request $(\boldsymbol{s}', \boldsymbol{x} + \boldsymbol{d}, \boldsymbol{d}, l)$ where $\boldsymbol{s}' = C(X[\boldsymbol{x}, l].\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d})$ was enqueued. Eventually in a further iteration of the main loop, the request was extracted from the queue and the state $\boldsymbol{s}'$ either found or stored in $X[\boldsymbol{x} + \boldsymbol{d}]$ at some level l''. In the same iteration at line 20 the destination level $X[\boldsymbol{x}, l].D[\boldsymbol{d}]$ was set to the correct value l'' then. $\square$

In the real implementation two additional problems have to be overcome:

- We work with finite precision floating point arithmetics. Therefore the inequality test at line 12 has to be implemented in a little more relaxed fashion. Instead of strict vector equality we use an application-specific *tolerance function* making the decision whether two state vectors should be considered as equal.
- Usually we are not sure whether a particular conversion function describing behaviour of the modeled virtual environment accomplishes the finiteness requirement. To prevent infinite computation we impose a fixed limit on the number of levels. If it is exceeded the program aborts. Such a critical condition indicates either true divergence (i. e. breaking the finiteness requirement) or some other problems, e. g. too strict tolerance function and/or insufficient precision of the conversion function computation etc.

We implemented the described algorithm as a generic ANSI C program skeleton into which an arbitrary conversion and tolerance functions can be plugged. Moreover, the current implementation exploits a fairly straightforward potential parallelism in processing the queue. However, detailed description of those features is beyond the scope of this article. Properties of the parallel algorithm are subject of current evaluation and will be reported elsewhere.

2.4. State Space Traversal

Given the precomputed 3D grid with conversions described in previous sections the haptic device can be driven in the real time without further need of time-consuming model evaluations. In this section we assume the simpler case that it is possible to compute both force interaction and visual representation of the model from a state vector sufficiently fast. If not, the required data has to be stored in the precomputed grid and the algorithm described in this section interpolates those data instead of the system state.

Algorithm 2 Haptic device driving

```
 1: $\boldsymbol{x} \leftarrow$ read HIP position
 2: initialize current cell w. r. t. $\boldsymbol{x}$
 3: loop
 4:    $\boldsymbol{x} \leftarrow$ read HIP position
 5:    if $\boldsymbol{x}$ not in current cell then
 6:       switch cell
 7:    end if
 8:    $\boldsymbol{s} \leftarrow$ interpolate states in cell corners w. r. t. $\boldsymbol{x}$
 9:    compute and apply the force
10: end loop
```

Skeleton of the driving program is shown in Algorithm 2. The program keeps track of the current cell of the grid where HIP is located. For each of the eight corners of the cell a *current level* and *current state* is maintained. For a triplet of indices $i, j, k \in \{0, 1\}$ representing a corner we denote $l[i, j, k]$ the current level and $\boldsymbol{s}[i, j, k]$ the current state in the corner.

The initialization steps involves finding the grid cell into which the starting HIP position $\boldsymbol{x}$ belongs, setting current level to 0 in all cell corners, and reading the precomputed states at level 0.

In each step of the main loop (running at the rate required to drive a haptic device, i. e. at least 1 kHz) the

HIP position is read from the device and mapped to the precomputed grid. As long as it stays within a single cell the current state vector is linearly interpolated among all the corners w. r. t. the states stored in the grid at the current corner levels. More precisely, let's denote $\bar{\boldsymbol{x}} = (\bar{x}_1, \bar{x}_2, \bar{x}_3)$ the normalized relative HIP position within the current cell. We compute corner weights as products

$$\begin{aligned} w[i,j,k] = &\big((1-i)\bar{x}_1 + i(1-\bar{x}_1)\big) \\ &\cdot \big((1-j)\bar{x}_2 + j(1-\bar{x}_2)\big) \\ &\cdot \big((1-k)\bar{x}_3 + k(1-\bar{x}_3)\big)\,. \end{aligned}$$

It can be seen that $\sum_{i,j,k\in\{0,1\}} w[i,j,k] = 1$ for each $\boldsymbol{x}$ within the cell. The interpolated state w. r. t. $\boldsymbol{x}$ is computed as a weighted sum

$$\bar{\boldsymbol{s}} = \sum_{i,j,k\in\{0,1\}} w[i,j,k]\,\boldsymbol{s}[i,j,k]\,.$$

It can be easily seen that $\bar{\boldsymbol{s}}$ becomes $\boldsymbol{s}[i,j,k]$ for $\bar{\boldsymbol{x}} = [i,j,k]$. Moreover, if a component of $\bar{\boldsymbol{x}}$ is 0 or 1, the weights of all opposite vertices become 0. In other words, for computation of an interpolated state vector at each point on the cell boundary (face or edge) only the precomputed state vectors in the vertices of the face or edge are considered.

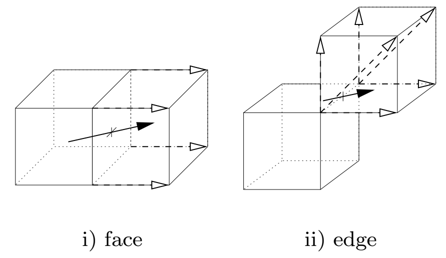
Eventually, the HIP crosses a cell boundary. In such a case two consequent measurements of HIP position fall into two distinct cells. Therefore the current cell has to be switched w. r. t. the new HIP position. Further on we assume that the two cells share at least one common vertex. If it is not true the HIP motion is split with interpolation into two or more steps for which the condition holds.

On a cell switch we preserve levels (and consequently state vectors) in all common vertices of the two cells. However, it is necessary to find out levels to be considered in the remaining vertices of the new cell. In order to choose an appropriate strategy we classify the cell switches in the following way (see also Fig. 3):

i) *Across face.* The old and new positions fall into cells which share a common face. In this case conversions from the common grid points in the direction perpendicular to the common face (i. e. along an axis) are considered.
For example, assuming the common face consists of vertices $[1,*,*]$ of the original cell (i. e. the vertices $[0,*,*]$ of the new cell) the only considered HIP position shift vector is $\boldsymbol{d} = (\bar{d},0,0)$. E. g. if the vertex $[1,0,0]$ of the original cell maps to a grid point $\boldsymbol{y}$ and a level l then the vertex $[0,0,0]$ in the new cell will map to the same point $\boldsymbol{y}$ and level l. The vertex $[1,0,0]$ in the new cell will map to $\boldsymbol{y}+\boldsymbol{d}$ and its level will be assigned the value $X[\boldsymbol{y},l].D[\boldsymbol{d}]$.

ii) *Across edge.* The old and new cells share a single edge. Three conversions from each of the two common vertices are considered, along the edges of the new cell as well as its face diagonals.

iii) *Across vertex.* The old and new cells share a single point. All seven conversions from the common point to the other vertices of the new cell are considered.



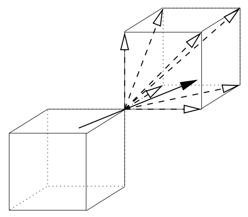


Figure 3: *Three ways of switching the current cell. Change of HIP position is shown with the black arrow, considered conversions are the white dashed arrows.*

As for each point on a cell boundary only the vertices of the respective edge of face are considered in the state vector interpolation, at any common point of the two switched cells the resulting state vector is the same, not regarding in which cell it was actually computed. Consequently, *the mapping of measured HIP position to the resulting state vector is always continuous.* This is a critical property when driving a haptic device.[6]

The cases ii) and iii) in the above list document the requirement to compute conversions also in the diagonal directions. Attempting to substitute two or three axes-aligned steps in place of a diagonal one we might face ambiguity — the resulting levels of the final vertex may differ when different possible paths were considered. Hence a direct model evaluation (represented by the diagonal steps) has to be used instead.

Simultaneous visualization of the model is done in an independent thread of the program. The visualization loop runs at much slower rate (about 25 Hz) reads the current interpolated state vector (shared with the haptic loop), and computes and displays visual representation of the model.

3. Flexible Molecular Models

In this section we briefly present an application area we are focused on, and show how the general method described in the previous section is deployed.

Molecular flexibility is a chemical property that usually becomes important when biological activity of molecules is considered. Flexible molecules are those which are able to change their shape quite easily, without changing their chemical structure, i. e. creating or splitting a bond. Capability of a molecule to achieve a particular shape may determine possibility of certain reaction. Protein synthesis and enzymatic reactions are good examples.

Properties of flexible molecules are described in terms of *conformational behaviour*. By a conformation we mean a distinct, relatively stable shape of the molecule (i. e. a shallow potential energy minimum). Conformational behaviour is the process of traversal among various conformations.

Conformational behaviour analysis is a well established area of computational chemistry. Since 80's numerous results were achieved with the aid of growing computational power, and huge amount of data was produced.

We claim that haptic interaction can considerably increase the quality and speed of human perception of those data. Conformational behaviour, like many other chemical processes, is determined by energetic constraints. On the other hand, with perception of a force the human is more aware of the energy of the examined system, as well as changes of the energy. Hence the haptic interface is capable to deliver such information to the user more directly compared to conventional methods.

In our prototype application we represent the molecule with its van der Waals surface — each atom is shown as a colored ball, and the balls are big enough to overlap one another. The user manipulates a *probe*, another ball virtually "attached" to the haptic device (Sensable's PHANToM). The probe and the molecular surface are not allowed to penetrate each other. Instead, the molecule is forced to change its shape according to the rules given by the underlying computational model of the conformational behaviour. The user "feels" the energy required to do the particular conformational change as a force feedback delivered by the device, as well as sees the effect of the applied force in a synchronous visualization of the model.

Our application considers a single conformation (minimum of potential energy of the molecule) and its local surroundings represented by several *transition states* — other shapes which correspond to directly accessible saddle points of the potential energy. For the sake of simplicity the reader can think of the transition states as shapes resulting from "twisting" the molecule along one or more bonds.

Given the shape of the conformation and a transition state, and a morphing parameter $t \in [0, 1]$ we are able to compute an intermediate shape such that the shape changes continuously from the conformation ($t = 0$) to the transition state ($t = 1$).[2] Moreover, the morphing algorithm can be modified so that it considers more transition states as well as a corresponding vector of the morphing parameters. The resulting shape is a merge of the shapes emerging from individual conformation to transition state paths. Hence the central conformation and the several transition states can be understood as generators of a certain space of shapes of the molecule. The morphing parameters become coordinates of a given shape within this space.

In terms of Sect. 2 we describe the entire model with a state vector. It is composed of variables of two sorts:

- morphing parameters (as many as the considered transition states),
- rotation and shift from an original position (6 variables in total).

The morphing parameters describe uniquely a shape of the molecule, the others allow the molecule to rotate and move.

We introduce a concept of *hybrid energy* (to be described in more detail elsewhere). This is a virtual quantity which unifies together:

- modeled property of the molecule, i. e. the energy required to perform a shape change,
- behaviour of the virtual environment and rules of the users' interaction with it.

Potential energy of the molecular shape, as derived from the underlying chemical calculations, is included directly in the hybrid energy formula. The user's interaction is expressed with additional terms which strongly penalize penetration of the molecular surface with the probe, and others which stabilize the molecule more or less at its initial position.

Given the energy, the principal physical law which determines model behaviour is the 2nd thermodynamic law — being given some degrees of freedom any natural system follows its energy gradient spontaneously until it reaches the closest local energy minimum. Presence of the 2nd thermodynamic law is ubiquitous in the real world. Consequently, its presence in a virtual environment helps the user to overcome the gap between her real-world experience and the virtual world behaviour. Then, she can concentrate on perceiving the information the virtual model was designed to deliver. Our experience shows that in this way we achieve a high degree of intuitiveness.

Within this framework, we interpret the probe position as an external constraint and let the hybrid energy continuously minimize, treating the state vector as free variables.

It is the hybrid energy minimization which represents most of the time consuming calculations in this application. Therefore, according to the general framework of Sect. 2, it has to be covered by the conversion function. Evaluating the conversion function $C(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d})$ for a feasible state $\boldsymbol{s} \in S(\boldsymbol{x})$ (i. e. $\boldsymbol{s}$ represents a shape of the molecule possible for the probe position $\boldsymbol{x}$) involves local minimization of the hybrid energy, starting in $\boldsymbol{s}$ and constraining the probe position to $\boldsymbol{x} + \boldsymbol{d}$.

As mentioned in Sect. 2.1 the state space calculation needs at least one seed point and its known state to start from. In our application we assume that a bounding box is specified in such a way that no interaction between the probe and the molecular surface occurs in its corners. Hence we seed the computation with the corners and a zero state vector — initial shape and position of the molecule.

Finally, the force function is expressed as a sum of *restorative forces*[6] — if the probe penetrates a surface of an atom they are repelled from each other with a force proportional to the amount of penetration.

Our experiments show that the calculation of an actual position of all atoms based on a state vector value as well as evaluating the force interaction is simple enough to be run within the haptic loop.

It can be shown that the force formula corresponds to a space gradient of scalar field of the hybrid energy corresponding to the probe positions and the respective system states. Again, this property meets intuitive expectations on the model behaviour — in the real world, a mechanical force is exactly the space gradient of a potential energy.

4. Results and Conclusion

The ideas presented in this paper are currently reflected in a prototype application. Up to now it has been tested with data on alanine amino acid (22 atoms) and a simple peptide (77 atoms).

The grid calculation is implemented in a distributed way, it's been successfully run on more than 100 CPUs. We are preparing precise measurements of scalability of the parallel algorithm. The conversion function is based on our molecular-shape morphing algorithm Aida[2] which in turn uses results of the Cicada[4] family of programs used for computational conformational analysis. However, the system is designed in such a way that it should be possible to plug-in any force field (conversion and force function) satisfying the described constraints.

The interactive part of the application is implemented in Linux OS, driving the PHANToM force feedback device, and visualizing the model through OpenGL.

Given the resolution and space extent of our device (the smallest 1.0 model) we found out experimentally that 30^3 grid points are sufficient to create illusion of a perfectly continuous model of the simpler molecule (alanine), and it does not make sense extend the grid beyond 50^3 for the other one. Table 2 gives an insight into absolute numbers of output size and CPU time of the grid computation.

Molecule	avg. levels	data size	CPU hours
alanine	1.023	5.2 MB	11
peptide	1.035	5.2 MB	27

Table 2: *Computation of the state space grid of the two testing molecules. Done on 700 MHz Pentium III processors. The grid resolution was 30 in both cases. The 2nd column shows an average number of distinct levels per grid point. Output data size refers to a Berkeley DB file.*

Up to now we performed too few computations to derive exhaustive results. However, the most important observation is the actual number of levels per grid point. The low numbers indicate that the phenomenon of multiple distinct states per HIP position, despite crucial from the qualitative point of view, is very rare when counted quantitatively. From practical viewpoint it is a positive result — we can expect that the method will be able to deal with much larger and more complex problems while still keeping affordable data size and computation time.

The application was presented to a group of users with knowledge in the field of computational chemistry but without prior experience with haptic computer driven devices. After a very short training (only a few minutes) the users did not find difficulties in interaction with the model. The received feedback was positive, the users claimed that the presented interactive model helps them understand conformational behaviour of the molecule much faster than conventional methods.

We consider the principal achievement of the work presented in this paper to be the design of a framework and related generic algorithms which can be used to build haptic interactive models of complex force fields. The strict separation of the computationally extensive calculation of the force field properties into the off-line phase makes the approach virtually independent on the force field calculation complexity.

Acknowledgement

This work was supported by the Grant Agency of the Czech Republic, grant no. 201/98/K041, and the Czech Ministry of Education, research programme CEZ:J07/98:143300003. Special thanks to the team of the HCI laboratory at the Faculty of Informatics for providing the application testing environment, the Supercomputer Centre Brno and the MetaCentre project for providing computing resources, and prof. Luděk Matyska for valuable feedback in preparation of this manuscript.

References

1. Z. Kabeláč. Rendering stiff walls with phantom. In *Proc. 2nd PHANToM User's Research Symposium*, 2000. `http://www.vision.ee.ethz.ch/~purs2000/Final_PS/Kabelac.pdf`.
2. Aleš Křenek. An algorithm on interpolating between two shapes of molecule. In W. Straßer, editor, *Proc. SCCG '97*, pages 77–84. Comenius University, Bratislava, 1997.
3. Aleš Křenek. Haptic rendering of molecular flexibility. In M. Harders and S. Huber, editors, *Proc. PURS 2000*, pages 19–26, 2000. ISBN: 3-89649-579-8.
4. J. Koča. Traveling through conformational space: an approach for analyzing the conformational behaviour of flexible molecules. *Progress in Biophys. and Mol. Biol.*, 70:137–173, 1998.
5. Maple, a symbolic mathematics system. `http://www.maplesoft.com`.
6. W. R. Mark, S. C. Randolph, M. Finch, J. M. Van Verth, and R. M. Taylor. Adding force feedback to graphics systems: Issues and solutions. In *Proc. SIGGRAPH*, 1996.
7. D. C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. In *Proc. SIGGRAPH*, pages 345–352, 1997.
8. C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. In *Dynamic Systems and Control*, volume 1, pages 146–150, 1994.