SOFA: Software for Observing Force-feedback Algorithms

Category: Research

ABSTRACT

Over the past few years, haptic algorithms have advanced considerably. However, no standard evaluation method exists, which allows researchers to compare their results in an objective manner. This paper elaborates on an evaluation method for haptic algorithms. As this method makes use of user input, an experiment was conducted to define how to the evaluation method should be applied.

In order to allow researchers to apply the evaluation method, a C++ library was developed: SOFA, Software for Observing Force-feedback Algorithms. This library can be used in haptic rendering libraries, which allows developers to provide their own haptic device implementation.

Finally, the SOFA library can also aid in debugging haptic algorithms, as it can provide developers with consistent haptic input and can allow them to experience what the users felt.

CR Categories: D.2.4 [Software Engineering]: Software/Program Verification—Validation; D.2.5 [Software Engineering]: Testing and Debugging—Debugging aids; H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces—Haptic I/O

Keywords: haptics, algorithm evaluation

1 INTRODUCTION

Although the number of haptic algorithms have grown over the last couple of years, no standards currently exist for evaluating and comparing haptic algorithms. Different methods have been used in the past. For instance, the time complexity of a haptic algorithms can be mathematically calculated [self-reference]. Although this leads to a better understanding of the algorithms' performance, it does not allow for the comparison of two algorithms with the same time complexity. Furthermore, due to the behaviour of the end user, some optimizations are difficult to predict. Thus, these theoretical findings should be complemented with real-life measurements in order to know the exact behaviour.

Another approach is to load the haptic loop until the calculations cannot be performed any longer in real-time. For instance, Acosta and Temkin [1] compared different versions of the GHOST API by rendering objects and scenes of increasing complexity until the API could not obey the 0.9ms maximum haptic rendering time. Although, it does not take the performance at a lower load into account, this approach is suited to test the limits of an implementation, which is a valuable metric.

As a third approach, a graphical tool can be used in order to visualize the hapic load. The GHOST haptic API provides a graphical tool which displays the haptic load using 10% intervals, as depicted in figure 1. This approach was used to compare the mesh rendering implementation of GHOST and e-Touch [2].

The above-mentioned approaches have two flaws. A first problem is that they do not provide exact numerical results, which can make it more difficult to draw conclusions. The second problem is that the algorithms are not compared in an equal manner. As the evaluation occurs while the user is interacting in real-time with the

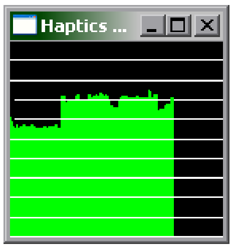


Figure 1: GHOST haptic load tool

algorithm, it is possible to unintentionally give one algorithm input which can be more easily handled (e.g. by unintentionally avoiding concave parts of the object used).

In this paper, we introduce the SOFA library, a C++ library, which supports an empirical approach for evaluation of haptic algorithms. In the next section, we will discuss this evaluation method. Afterwards, we elaborate on the SOFA library and explain how it can support the evaluation method. Next, an experiment will be described in which we investigated how the evaluation method can be applied. Finally, we draw some conclusions.

2 EVALUATION METHOD FOR HAPTIC ALGORITHMS

The evaluation method is based on our theoretical framework for comparing haptic algorithms. In this section, we will summarize the major points of this framework. For a more detailed and formal explanation, we refer the interested reader to [self-reference]. We will first give a definition of the terms, our evaluation method is based on.

2.1 Defintions

First, we define what we mean with haptic algorithms in the scope of this paper.

Definition 1 *A* geometric haptic algorithm *is a two-fold algorithm. Its input is the current position and velocity of the pointer, as defined by the force-feedback device, and a virtual object , which has to be rendered. The first part is a collision-detection step, which calculates whether the pointer position is located inside the object. The second step, called the render step, calculates the surface contact point (SCP) and the force that should be exerted by the force-feedback device.*

Next, we can define if two algorithms produce the same output.

Definition 2 *Two haptic algorithms are* collision equivalent *if their collision-detection steps produce the same result for each possible input.*

Definition 3 *Two haptic algorithms are* render equivalent *if they are collision equivalent and the position of the SCP, as calculated in their render steps, differ less than the appropriate JND*[1] *for each possible input.*

[1]The just noticeable difference (JND) is the smallest change in pressure, position, ... that can be detected by a human and depends on the body

Definition 4 *Finally, two haptic algorithms are* equal *if they are render-equivalent and the forces, as calculated by their render steps, differ less than the appropriate JND for each possible input.*

We define the three levels in definitions 2 through 4 because different algorithms can have different purposes. For instance, a new algorithm can be created in order to be faster. In that case, the new algorithm should be equal to existing implementations.

It is also possible that a new algorithm modifies the force vector in order to implement a haptic texture [8]. This algorithm can be render-equivalent to existing algorithms.

Finally, a new algorithm could be created which rounds certain edges by modifying the SCP. This algorithm can be collision equivalent to existing algorithms.

2.2 Evaluation Method

The evaluation method consists of four steps. In a first step, a number of users inspect a number of 3D objects with a haptic device for a certain amount of time. During this phase, the position and velocity of the user's pointer is recorded in the haptic loop.

In a next step, this data is used as input for the algorithms that are being evaluated. This means that the stored user input is used instead of real-time user input, thus ensuring that all algorithms receive the same input values. The time needed to execute each haptic loop, the SCPs and the calculated forces are logged.

Next, the logged data is stored in a database for easy retrieval. This is needed in order to handle the large amounts of data involved. For instance, the experiment in section 4 resulted in 1,440,000 measurements.

Finally, the data can be analyzed. When comparing two algorithms, we can not only use statistical techniques in order to compare which one is faster, but we can also see if they are collision-equivalent, render-equivalent or equal. This should be checked, as this evaluation method only works on algorithms that are at least collision-equivalent.

When comparing the execution times of the different algorithms, one must make sure to not just take the average time of the different loops. A loop in which the pointer collides with the object will take more time, as the SCP and the force have to be calculated. It is therefore better to calculate the average time of the loops where no collision has occurred and the average time of the loops with collision.

This evaluation method has been put to the test by evaluating a naive and an optimized algorithm, as these results can be easily interpreted. We found that the results of our evaluation method are supported by the theoretical comparison of both implementations [self-reference]. It was also successfully used for the evaluation of different algorithms for haptic cloth rendering [self-reference].

However, some question do remain about the exact usage of the evaluation method, which will be explained in the next section.

2.3 Open Questions

As it is impossible to provide the algorithms with all possible input combinations, we have to limit the amount of data that is collected in the first step. Assuming that n_u users inspect n_o objects for t_e seconds and that the haptic loop has a sampling rate of f_{sr} Hertz, we obtain the following number of data points n_d:

$$n_d = n_u \times n_o \times t_e \times f_{sr} \tag{1}$$

area where the stimulus is applied [3]. As an example, one cannot tell the difference between two orientations of one's wrist if they are less than 2.5° apart.

The only constant in this equation is f_{sr} as most haptic APIs use 1000 for this value. However, the question remains how many objects should be used, the number of test subjects and the amount of time that they can investigate the objects. Furthermore, objects can range in complexity. Here we can differentiate between complexity for the algorithm (e.g. the polygon count) and for the user (e.g. different geometric features). Furthermore, users with a different experience level do not use haptics in the same manner. Usability tests of interaction paradigms have shown us that user experience can play a major role in the evaluation [self-reference]. This can potentially pose a problem, as people with much experience are hard to find. Besides the researchers involved in developing the algorithm to be tested, it is difficult to find people who are familiar enough with haptics. Of course, the involved researchers should not participate in the experiment as they may involuntary influence the outcome of the result because of their knowledge of the system to be tested.

Finally, the user must have enough time to be able to fully inspect the object, but this should not take to long in order not to bore them.

Section 4 will elaborate on an experiment we conducted in order to answer some of these remaining questions. First, we will explain the SOFA library, which can be used to support the evaluation method.

3 Software for Observing Force-feedback Algorithms

The evaluation method described in this paper is supported by the SOFA library, Software for Observing Force-feedback Algorithms. SOFA consist of a C++ library, which can be incorporated in most haptic libraries, and a number of Perl scripts, which can extract data from the database in which the SOFA C++ library stores its results. For this database PostgreSQL, a powerful open source database, is used.

This section explains how SOFA can be used in the four steps of our evaluation method, as explained in section 2.2, and discusses a Haptic Viewer which can aid in the analysis and discussion of haptic algorithms. The SOFA library and Haptic Viewer are available on http://... (will be made public if the paper is accepted).

3.1 Recording

One of the classes in the SOFA library allows to log the user's position and velocity. This *Recorder* class provides a method which should be called in each loop. This method assigns an ID to the current loop and records the pointer's position and velocity, which it receives as parameters.

The *Recorder* class stores the positions and velocity in main memory as disk latencies could result in an unacceptable slowdown of the haptic loop. Therefore, the *Recorder* class also provides a method for saving the log to a comma-separated values file afterwards. This log file can than be used in the next step of the evaluation method. Listing 1 shows the first lines of an example log file.

The first line indicates that this is a file that can be handled by the SOFA library. The following lines each contain the ID of the loop, the pointer position and velocity. Note that this velocity is 0 in these lines, as this is the result returned by the PHANToM device; after a few loops, the velocity gets a non-zero value.

Listing 1: Example SOFA recording

```
Sofa recording v1.0
0,0.00189182,−0.0044273,−0.00369428,0,0,0
1,0.00189262,−0.00295216,−0.00221647,0,0,0
2,0.00126255,−0.00295279,−1.37262e−006,0,0,0
3,0.00126201,−0.00295258,−0.00147804,0,0,0
```

The *Recorder* class can be used in each haptic library, which can be extended (e.g. GHOST). The researcher wanting to use the evaluation method has to write an application, which allows the user to investigate the object. This application hence has to provide graphic feedback and haptic feedback using a reference algorithm, and must use the *Recorder* class to log the haptic data. Listing 2 gives an example of the *Recorder* class' use.

Listing 2: Example use of the *Recording* class

```
// create a recording object
Sofa::Recorder recorder;

// record the information to main memory
recorder.Start();
while ( /* recording */ )
{
    /* get current position and velocity
       from the haptic device */
    recorder.Log( pos_x, pos_y, pos_z,
        vel_x, vel_y, vel_z );
}
recorder.stop();

// save the information to the disk
recorder.Save( logfile );
```

3.2 Playback

In order to provide the data saved by the *Recorder* class, the SOFA library provides a *Playback* class, which can read back the comma-separated file and provide the data to an application. However, the data used does not have to be limited to prerecorded data. Although, we advise to use data from user interactions, it is also possible to use other data, such as recordings of physical objects [10], in order to validate the correctness of the algorithms, as long as the data is stored using the correct file format.

The *Playback* class can only be used in a haptic library that allows the use of different haptic devices, such as CHAI 3D [5], e-Touch [2], Haptik [6] or HAL (see section 3.5). In order to use this class for the evaluation, an application must be written, which uses one of the algorithms to be tested and a *pseudo* haptic device, which acts like a real haptic device, but uses the data from the comma-separated file instead of life interactions with a user. This application must use the same objects as the application that recorded the user data. Listing 3 gives an example of the *Playback* class' use.

This process is not only suitable for evaluating haptic algorithm, but can also used for debugging purposes. As haptic applications are highly interactive, it is difficult to reproduce the exact user input that led to a problem. It is also often difficult to use breakpoints while debugging haptic applications due to their highly interactive nature. By logging the user input, developers can recreate the problem time after time.

3.3 Logging

Each loop, the data provided by the algorithm to be tested has to be stored. A third class, the *Logger* class, provides this functionality. This class contains methods for measuring the time needed to execute the algorithm, using the MS Windows high performance counters. These methods for starting and stopping the measurement should be called just before and after the executing of the algorithm in order to avoid measuring other parts of the haptic loop (e.g. the time needed to traverse the scene graph). Since MS Windows is not a real-time operating system, unexpected delays can occur [4]. This can be minimized, by setting the measurement thread's priority to "time critical" and by allowing other threads to be executed during each loop just before the algorithm is measured. Furthermore, the measurement thread should always execute on the same processor if a multi-processor system is used as a switch from one processor to another has an influence on the high performance counters.

The *Logger* class must also be called in each loop in order to store the results of the algorithm to be tested in the PostgreSQL database. The class provides a method for this purpose, which stores the IDs of the algortihm, test subject, object and loop. Furthermore, the fact if collision has occurred, the SCP and the force vector are stored in the database using the C library which is provided with PostgreSQL. Listing 3 gives an example of the *Logger* class' use.

Listing 3: Example use of the *Playback* and the *Logger* classes

```
// create a playback object
Sofa::Playback playback( logfile );

//create a logger object and connect to the DB
Sofa::Logger logger( algorithm, subject, object );
logger.Init( "localhost", "sofa", "sofa", "sofa" );

// read the data from the log file
while ( playback.NextLoop() )
{
    playback.GetPosition( pos_x, pos_y, pos_z );
    playback.GetVelocity( vel_x, vel_y, vel_z );

    SwitchToThread();
    logger.Start();
    /* give the data to the haptic algortihm */
    logger.Stop();

    logger.Log( loop, collision, SCP_x, SCP_y,
        SCP_z, force_x, force_y, force_z );
}

// let the DB commit the data
logger.SaveResults();
```

The *Recorder* class stores its data in comma-separated files for performance purposes. However, for the analysis phase, it is more efficient to also have this data in the database. SOFA therefore provides a Perl script for storing the *Recorder* data into the database.

3.4 Analysis

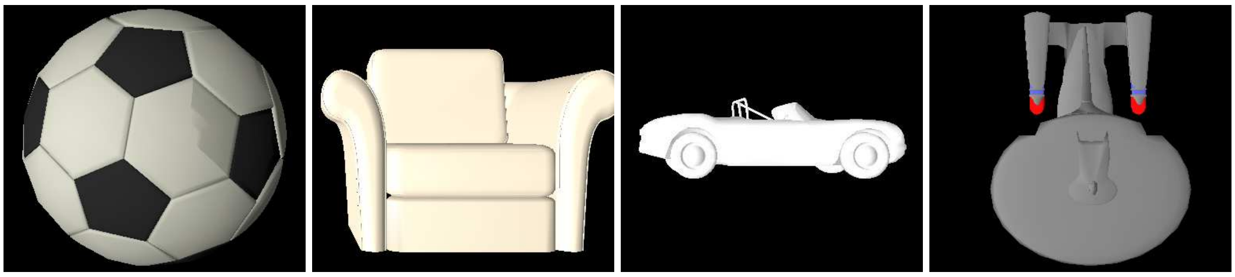
After all data is logged, the database can support in the analysis phase. An administration program, such as pgAdmin III, which comes with PostgreSQL and allows the user to make SQL queries and copy the result to another program, can be used. For instance, the results of table 1 were obtained by executing the SQL query in listing 4.

Listing 4: Sample SQL query

```
SELECT subject, STDDEV(position_x),
    STDDEV(position_y), STDDEV(position_z)
FROM logs GROUP BY subject
```

However, SOFA already has a number of Perl scripts which can be used for the analysis. These scripts make use of a common Perl module, which handles the communication with the database and is part of SOFA. It is thus easy for researchers to write their own Perl scripts for performing an analysis that is specific for their research.

One of these scripts checks if the tested algorithms are collision equivalent as this is the requirement for our evaluation method. A



(a) Sphere with a football texture (b) Couch (c) Car (d) USS Enterprise

Figure 2: Objects used in the experiment (curtesy of www.3dcafe.com)

second Perl script indicates in which loops the tested algorithms return a different result for the collision detection step as the previous loop. This can help to understand the behaviour of the algorithms.

The third Perl script exports the time measurements as two files that can be read by statistical programs, such as R [9]: one file for the loops where no collision occurred and one files for the loops where collision occurred. This way the algorithms can be statistically compared. SOFA also contains an R script for executing the statistical analysis. However, a number of statistical packages have problems handling the large amount of data involved.

3.5 Haptic Viewer

A last Perl script extracts data from the database into comma-separated file, which allows developers of an haptic algorithm to recreate the feeling that a test user had. This data combines the data stored the *Recorder* by class and by the *Logger*, as can be seen from listing 5.

Listing 5: Example input for the Haptic Viewer

```
Sofa viewing v1.0
0.00189182,-0.0044273,-0.00369428, ↩
       0.001892,-0.004427,-0.003694,0,0,0
0.00189262,-0.00295216,-0.00221647, ↩
       0.001893,-0.002952,-0.002216,0,0,0
0.00126255,-0.00295279,-1.37262e-006, ↩
       0.001263,-0.002953,-1e-006,0,0,0
0.00126201,-0.00295258,-0.00147804, ↩
       0.001262,-0.002953,-0.001478,0,0,0
```

Each line consists of the pointer position, the SCP and force in a particular haptic loop. As no collision occurs in this example, the SCP is equal to the pointer position and the force is zero.

Whereas in training systems, such as the Virtual Haptic Back [11], forces are used to draw the user towards the recording path, the haptic viewer returns the recorded forces to the user. The original position and the SCP is visually rendered as a red and blue sphere respectively. Furthermore, the force is depicted by a line, starting from the SCP and in the direction of the force vector. This can be useful for discussing rendering artifacts, since these can be felt in the same manner by all parties involved.

The haptic viewer can also be useful to communicate the results of an algorithm to fellow researchers. This has already been achieved for inspecting objects. Phantom-X [7] allows users to feel objects via an ActiveX control in their web browser. Although, this has as advantage that the user is able to see the original object, this ActiveX control only supports a limited number of algorithms and devices.

The SOFA haptic viewer makes use of our haptic library, HAL. This library consists of a extensible core library and a number of libraries, which implements haptic devices, scene graphs and object rendering algorithms. The haptic device can be loaded by the HAL library at run-time. This means that an application, such as the Haptic Viewer, that is written on top of HAL does not need to know the implementation of the haptic device.

The next section elaborates on an experiment that was conducted using the SOFA library.

4 Experiment

In order to address the problems as elaborated on section 2.3, we conducted an experiment in which different test subjects were asked to interact with four different virtual objects. These objects, as depicted in figure 2, each have a different complexity, both for the algorithm as for the test subject.

For this experiment, we choose 12 test subjects: four persons with much experience with haptics (haptics researchers), four persons with moderate experience (who have occasionally used a haptic device) and four persons with little to no experience. All twelve test subjects inspected all four objects for 30 seconds. In order to avoid learning effects, we used a latin square design in order to to present the four objects to each test subject within one of the three groups in a different sequence.

We recorded and logged for each loop the position and velocity of the pointer, the fact if collision has occurred, the SCP and the output force. This was realized by using the SOFA library for recording, logging and analysing the results, and CHAI 3D [5] as haptic library.

The results of the experiment are discussed in the next section.

5 Results and Discussion

In order to define a set of recommendations for evaluating haptic algorithms, we investigated the influence of user behaviour and object complexity.

5.1 Spread of Pointer Position

In order to analyze how the test subjects investigated the virtual objects, we first looked at the spread of the pointer positions during the experiment, using the standard deviation of the pointers' X, Y and Z coordinates. In our work, we use a right-handed co-ordinate sytem, where X, is right-left and Y is up-down. As can be seen from table 1, little difference in standard deviation can be found among the test subjects.

More importantly, we wanted to investigate if there exists a difference between users with a different level of experience. We therefore, divided the test subjects into three groups according to experience level and calculated the standard deviation. Table 2 confirms that experience makes little difference in the spread of the pointer position.

Table 1: Standard deviation of the user's point position, grouped by user

user	X	Y	Z
1	0.44705	0.33562	0.32943
2	0.42455	0.30784	0.32465
3	0.43965	0.39586	0.34408
4	0.50323	0.39450	0.31595
5	0.41006	0.29739	0.27124
6	0.39574	0402010	0.42818
7	0.44907	0.38026	0.36373
8	0.47683	0.33691	0.34002
9	0.42557	0.43067	0.37193
10	0.49264	0.3430	0.41487
11	0.47935	0.3888	0.45041
12	0.46881	0.3258	0.46392

Table 2: Standard deviation of the user's pointer position, grouped by experience level

experience	X	Y	Z
much experience	0.46842	0.36178	0.33767
some experience	0.44935	0.35791	0.36159
little experience	0.46890	0.38318	0.43028

Although, little variation exists between test subjects, the complexity of the object being investigated does have an influence on the spread of the pointer movements. As can be seen from table 3, the shape of the object has an influence on the user's movements. We did not find any significance for the X values, but this can be explained as the user first had to move their pointer to the right since the objects were located to the right of the pointer's starting position. Using an F-test for variance we found a significant difference for the Y value between object 1 and the three other objects and between objects 2 and 4. We also found a similar results for the Z values.

Table 3: Standard deviation of the user's pointer position, grouped by object

object	X	Y	Z
1	0.50994	0.48185	0.49315
2	0.46394	0.31897	0.44135
3	0.49595	0.15784	0.25430
4	0.36212	0.41015	0.20122

5.2 Collisions

When looking at the number of collisions with the objects, as depicted in table 4, one can see that the test subjects were more confident in investigating object 1 (the football). Using paired student t-tests, we can find a significant difference between this object and the other objects ($p<.5$). No significant difference between the other objects can be found.

This implies that an evaluation of a haptic algorithm should incorporate both simple as complex objects. Using only simple objects will generate little data to validate the case were no collision occurs. Using also complex objects increases this amount of data. Combining both makes sure that enough data is collected when collision occurs as this will likely be more critical for the haptic algorithm. Furthermore, the test subjects indicated that the task of exploring the football was very simple. Also using complex objects helps to keep the test subjects engaged.

Table 4: Number of collisions per user and per object

user	object			
1	25617	17101	21374	20354
2	25648	17184	17843	18348
3	28774	21539	18449	15859
4	29448	17850	16477	15535
5	29385	17692	13895	16472
6	28015	11763	18897	14687
7	28668	19460	19632	19708
8	29159	16531	16645	15678
9	25263	17578	14770	14791
10	28783	17710	17887	10225
11	28939	13626	8577	14116
12	26489	15323	12843	12456

On the other hand, we can see a difference in the behaviour of test subjects according to the experience they have with haptics. Table 5 groups the total number of collision by experience level. Using paired t-test, we found a significant difference ($p<.5$) between users with much experience and users with little experience for objects 3 and 4. For object 4, we also found a significant difference between users with some experience and users with little experience

Table 5: Number of collisions per object, grouped by user experience

experience	object			
	1	2	3	4
much experience	109487	73674	74143	70096
some experience	115227	65446	69069	66545
little experience	109474	64237	54077	51588

This implies that an evaluation of a haptic algorithm should incorporate both test subjects with much experience as test subjects with little experience.

Furthermore, when we look at the transitions between collision and no collision and vice versa, we can see a significant difference between the first object and the other objects ($p<.001$). However, we cannot find a significant difference when looking at the experience level. Table 6 summarizes these results.

Table 6: Average number of transitions between collision and no-collision and vice versa, grouped by user experience

experience	object			
	1	2	3	4
much experience	16.5	52.25	89	76.75
some experience	11.25	77.75	51.75	41.25
little experience	20.5	48.5	57.5	55

5.3 Forces

When comparing the average of the forces that the algorithm had to calculate, little difference between the three categories of experience can be found, as shown in table 7.

5.4 Discussion

In this experiment, we obtained 1,440,000 data points, as can be seen by filling in values of equation 1:

Table 7: Average length of the force vector, grouped by user experience

experience	object			
	1	2	3	4
much experience	9.301	10.740	9.831	6.764
some experience	8.904	9.0184	10.043	6.466
little experience	8.723	7.940	7.7962	6.425

$$\begin{aligned} n_d &= n_u \times n_o \times t_e \times f_{sr} \\ &= 12 \times 4 \times 30 \times 1000 \\ &= 1,440,000 \end{aligned} \tag{2}$$

This amount of data is too large to be handled efficiently, considering that this has to be multiplied by the number of algorithms to be tested. Also, the setup of the test can be difficult. In this paper, we had 4 experienced participants as only 4 members of our research group (excluding the authors) met this criterium. Furthermore, finding good objects can be difficult at least, as most available objects are made with only visual applications in mind, thus having invisible gaps, where the user's pointer can "fall" through. This can be remedied if researchers make their test data public, but still provides difficulties when evaluation a new kind of haptic algorithms. For instance, the data presented in this paper is suitable for rendering algorithms that deal with rigid objects, not with deformable objects.

It is therefore advisable to reduce the amount of data. This can not be achieved by reducing the haptic sampling rate, as the 1 kHz rendering rate is an established value. Furthermore, during our experiment we did not find any reason that the 30 second sampling time was not suitable.

On the other hand, the small amount of difference between the complex objects suggests that the number of objects can be decreased. We propose to use one simple, such as object 1, one more complex objects, such as object 2 and one complex object, such as object 3 or 4.

Likewise, the number of test subjects can be decreased. We found a significant difference between test subjects with much experience and test subjects with little experience. Therefore, both categories should be involved. Within the groups, little differences can be found. We therefore propose to use two users with much experience and two users with little experience.

This leads us to a smaller amount of data as calculated in equation 3.

$$\begin{aligned} n_d &= n_u \times n_o \times t_e \times f_{sr} \\ &= 4 \times 3 \times 30 \times 1000 \\ &= 360.000 \end{aligned} \tag{3}$$

Although this amount of data is still considerable, it can be handled more easily.

6 Conclusions

This paper discussed a method for evaluating haptic algorithms in an objective manner. Based on our theoretical framework for the evaluation method, we investigated the sample size that is needed in order to perform the evaluation. Furthermore, SOFA, a software library for conducting such an evaluation was elaborated on. This library can be used in extensible haptic libraries. The experiment conducted not only determined the sample size, but also showed that SOFA can be used in combination with the CHAI 3D library.

References

[1] Eric Acosta and Bharti Temkin. Scene complexity: A measure for real-time stable haptic applications. In *Proceedings of the sixth PHANToM Users Group Workshop*, Aspen, CO, USA, October 27–30 2001.

[2] Tom Anderson and Nick Brown. The activepolygon polygonal algorithm for haptic force generation. In *Proceedings of the sixth PHANToM Users Group Workshop*, Aspen, CO, USA, October 27–30 2001.

[3] Grigore C. Burdea. *Force And Touch Feedback For Virtual Reality.* Winley Inter-Science, 1996.

[4] Nicolas Castagne, Jean-Loup Florens, and Annie Luciani. Computer platforms for hard-real time and high quality ergotic multisensory systems. In *Proceedings of 2nd International Conference on Enactive Interfaces*, Genoa, IT, November 17–18 2005.

[5] F. Conti, F. Barbagli, R. Balaniuk, M. Halg, C. Lu, and D. Morris. The CHAI libraries. In *Proceedings of Eurohaptics 2003*, pages 496–500, Dublin, IE, July 6–9 2003.

[6] M. de Pascale, G. de Pascale, F. Barbagli, and D. Prattichizzo. The haptik library: a component based architecture for haptic device access. In *Proceedings of Eurohaptics 2004*, pages 44–51, Munich, Germany, June 5–7 2004.

[7] Unnur Gretarsdottir, Federico Barbagli, and Kenneth Salisbury. Phantom-X. In *Proceedings of Eurohaptics 2003*, pages 466–470, Dublin, IE, July 6–9 2003.

[8] Marylin Rose McGee, Phil Gray, and Stephen Brewster. The effective combination of haptic and auditory textural information. *Lecture Notes in Computer Science*, 2058:118–126, 2001.

[9] R Development Core Team. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-07-0.

[10] Emanuele Ruffaldi, Dan Morris, Timothy Edmunds, Federico Barbagli, and Dinesh K. Pai. Standardized evaluation of haptic rendering systems. To appear in proceedings of 14th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Arlington, VA, USA, March 25–26 2006.

[11] Robert L. Williams, Mayank Srivastava, Robert Conaster, and John N. Howell. Implementation and evaluation of a haptic playback system. *Haptics-e, The Electronic Journal of Haptics Research (www.haptics-e.org)*, 3(3), May 3 2004.