Jakub Smolka[1]

EFFICIENCY OF DATA INTERPOLATION METHODS

Regardless of its type, algorithm efficiency can have a strong influence on a project's cost. This paper focuses on three algorithms used, among others, in computer animation. The algorithms are: blended parabolas, standard Catmull-Rom spline and modified Catmull-Rom spline. It briefly describes the algorithms and discusses differences in the implementations created by the author: one that doesn't depend on external libraries and another that uses Eigen. Each has two variants: one for 2D data and one for 3D data. The results of a numerical experiment are presented and discussed.
Keywords: *interpolation method, algorithm implementation, efficiency, performance.*
JEL codes: C63, C99.

Якуб Смолка

ЕФЕКТИВНІСТЬ МЕТОДІВ ІНТЕРПОЛЯЦІЇ ДАНИХ

У статті доведено, що незалежно від типу алгоритму його ефективність може мати сильний вплив на вартість проекту. Описано три алгоритми, які використовуються, зокрема, в галузі комп'ютерної анімації (метод суміщених парабол, стандартний сплайн Катмулла-Рома і модифікований сплайн Катмулла-Рома). Описано відмінності у варіантах їх реалізації, створених автором: один варіант – незалежний від зовнішніх бібліотек, і другий – з використанням бібліотеки шаблонів Eigen. Кожен з них має два варіанти: для 2D і для 3D-даних. Представлено результати чисельного експерименту та їх інтерпретацію.
Ключові слова: *метод інтерполяції, реалізація алгоритму, ефективність, продуктивність.*
Табл. 1. Рис. 10. Літ. 11.

Якуб Смолка

ЭФФЕКТИВНОСТЬ МЕТОДОВ ИНТЕРПОЛЯЦИИ ДАННЫХ

В статье доказано, что независимо от типа алгоритма его эффективность может оказать сильное влияние на стоимость проекта. Описаны три алгоритма, используемых, в частности, в области компьютерной анимации (метод совмещенных парабол, стандартный сплайн Катмулла-Рома и модифицированный сплайн Катмулла-Рома). Описаны отличия в вариантах их реализации, созданных автором: один вариант – не зависящий от внешних библиотек, и другой – с использованием библиотеки шаблонов Eigen. Каждый из них имеет два варианта: для 2D и для 3D-данных. Представлены результаты численного эксперимента и их интерпретация.
Ключевые слова: *метод интерполяции, реализация алгоритма, эффективность, производительность.*

Introduction

Interpolation is an important aspect in many data processing tasks. These include plotting graphs, economic data analysis, image processing and computer animation (Parent, 2008; Kopniak, 2010; Kopniak, 2011; Kopniak, 2012). Formally, interpolation is a tool for computing intermediate values between those from an initial discrete set of given data points. There are many and varied interpolation algorithms. They differ mainly in complexity, time consumption and accuracy of the results. Virtually all data of any numerical type can be interpolated. In computer animation it can be: an object's position in space or its orientation (angular position). The paper focuses on three algorithms: Blended Parabolas (hereinafter BP), standard

[1] PhD, Assistant Professor, Lublin University of Technology, Poland.

Catmull-Rom spline (hereinafter CM12) and modified Catmull-Rom spline (hereinafter CMP).

Economic aspect of algorithms

As stated above, interpolation algorithms differ from one another. Selecting an algorithm that suits a certain task is not easy. The nature of the input data and the desired results have to be taken into account. Not all algorithms work with certain input values and not all produce good results. Even if a proper algorithm is selected with respect to input data, complexity and result accuracy, there still remains one important aspect – time consumption. If a project is in its developmental stage, it is beneficial for the algorithms to be implemented in a way that allows for easy modification. When the project enters its production phase, they should be as efficient as possible even if their implementation is more complex. For example, the film studios that use computer animation to create special effects in movies use so called "render farms." These are clusters of computers that result in the quick rendering of finished movie frames. Such farms can consist of thousands of processor cores. For instance, a render farm that is used by the famous Industrial Light and Magic consists of 5700 cores (Tom's Hardware, 2013). Taking into account the fact that an average 4-core processor alone costs approximately 200 USD or the fact that rendering one frame can take up to 72 hours (Tom's Hardware, 2013), one can see that algorithm speedup by a factor of 2 or 3 times can have a significant impact on a hardware cost or the time required to finish the project. How versions of the same algorithm can differ with respect to speed is shown below.

Implemented interpolation methods

Interpolation methods can be expressed in various forms: geometric or algebraic. However, the most convenient form for creating computer implementation is the matrix form. It is given by the equations (Parent, 2008):

$$P(u)=U^T MB;\ U^T=[u^3\ u^2\ u\ 1], \qquad 1)$$

where: U – is the variable vector, u – variable in the range of [0; 1], M – coefficient matrix, B – vector containing geometric information (points or tangents).

The matrix form in eq. 1 is general enough to obtain different interpolation algorithms by simply exchanging the matrix M and vector B. Equations for the three implemented methods are given in Table 1.

With the equations given in Table 1, interpolated values are always generated on the curve between points P_i and P_{i+1}. The transition from P_i to P_{i+1} happens as the u parameter moves from 0 to 1. In the reminder of this paper this range between 2 points is referred to as the "interpolation range". This range is divided into "steps" and each step corresponds to $\Delta u=1/steps$ increase in the u parameter.

The above description is brief and given for the sake of completeness. More information on the above interpolation methods can be found in Parent (2008).

Properties of interpolation algorithms' implementations

Two different implementations of interpolation algorithms were created by the author, both in C++ (Eckel, 2002). The first one is based on class templates and is autonomous; it does not depend on any external library. The second one utilizes Eigen, a highly optimized library for linear algebra (Eigen, 2013; Guennbaud, 2013). These implementations are described below in more detail.

Table 1. **Implemented equations (P_i – coordinates of i^{th} data-set point)**

Method	M matrix	Point vector	Tangents
BP	$M=\frac{1}{2}\begin{bmatrix}-1 & 3 & -3 & 1\\ 2 & -5 & 4 & -1\\ -1 & 0 & 1 & 0\\ 0 & 2 & 0 & 0\end{bmatrix}$	$B=\begin{bmatrix}P_i\\ P_{i+1}\\ P_{i+2}\\ P_{i+3}\end{bmatrix}$	"built in" in M
CM12	$M=\begin{bmatrix}2 & -2 & 1 & 1\\ -3 & 3 & -2 & -1\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}$	$B=\begin{bmatrix}P_i\\ P_{i+1}\\ T_L\\ T_R\end{bmatrix}$	$T_L=\frac{1}{2}(P_{i+1}-P_{i-1})$ $T_R=\frac{1}{2}(P_{i+2}-P_i)$
CMP	$M=\begin{bmatrix}2 & -2 & 1 & 1\\ -3 & 3 & -2 & -1\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}$	$B=\begin{bmatrix}P_i\\ P_{i+1}\\ T_L\\ T_R\end{bmatrix}$	$T_L=\frac{\lvert P_{i+1}-P_i\rvert}{\lvert P_{i+1}-P_{i-1}\rvert}(P_{i+1}-P_{i-1})$ $T_R=\frac{\lvert P_{i+1}-P_i\rvert}{\lvert P_{i+2}-P_i\rvert}(P_{i+2}-P_i)$

Sourse: Calculated by the author.

The design goal of the first implementation is to make experimentation more straight forward. Since C++ standard lacks built-in matrix and vector algebra capabilities, these were implemented by the author. The implementation consists of several class templates that are shown on the UML diagram (Fowler, 2003; Podeswa, 2009) in Figures 1 and 2. The Matrix class template is an abstract description of a 2D matrix that can hold almost any data type. It also allows for matrix manipulation that performs typical matrix operations such as addition or multiplication in a natural manner. This is achieved by operator overloading. The Vector class template is derived from a Matrix and is designed to make vector manipulation more straight forward. The following class templates are more specific to the interpolation problem. The MMatrix is a convenience class derived from the Matrix class template. It contains coefficients from Table 1. The Point2D class template obviously represents a point in 2D space. Its overloaded operators make it compatible with the Vector class template (for example, subtraction of two points produces a vector). The Point3D class is omitted in the diagrams since it is almost identical to Point2D. The Tangents class template and its derivatives (CM12Tangents, CMPTangents) allow for computing required tangents based on a point containing data-sets.

As stated above, the design goal of this implementation is to make it easy to change and experiment with interpolation methods. Using this implementation for computing an interpolated value, is very straightforward: *result[i]=vecU.getTransposed()*matM*vecB*. The code closely resembles mathematical notation. This implementation will be referred to as "author's" in the reminder of this paper.

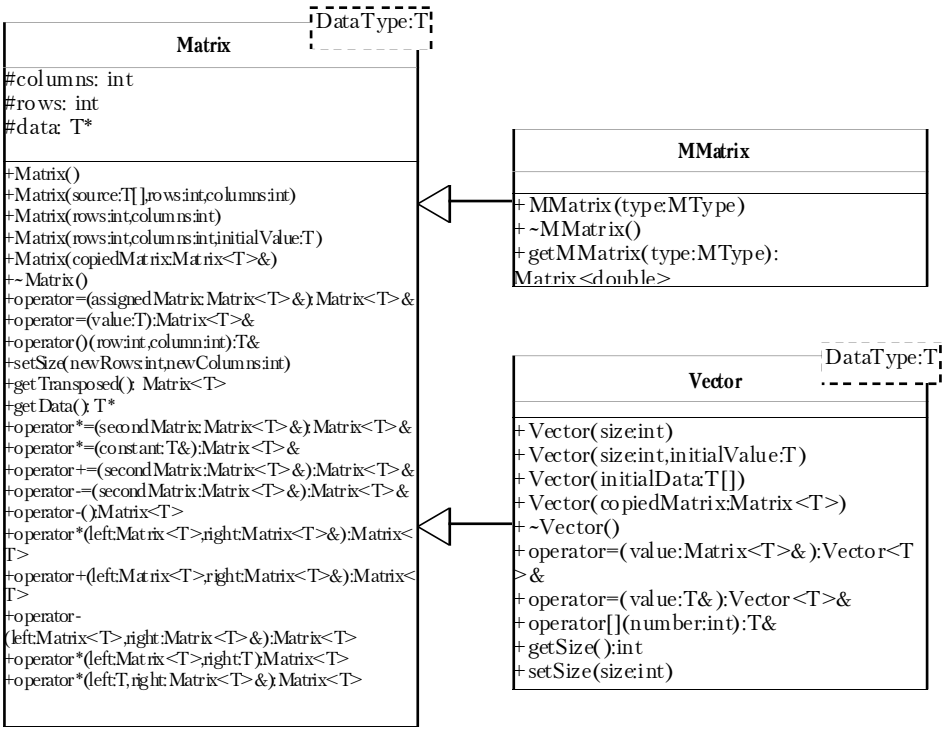

Figure 1. **UML diagram of the Matrix and Vector class templates** *(author's development)*

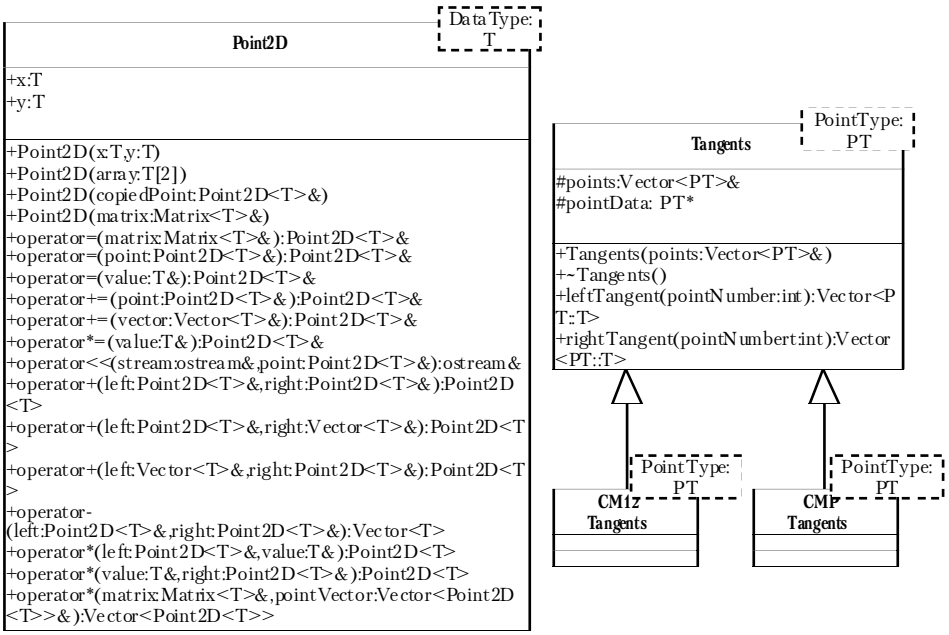

Figure 2. **UML diagrams of Point and Tangent class templates** *(author's development)*

The design goal of the second implementation is to make the code compatible with the b-tk library (Biomechanical Toolkit, 2013) (which is used by the author for motion capture data processing) and to increase the overall computation speed. Both these goals may be achieved by using the Eigen library (Eigen, 2013). The library supports matrices of different sizes and enables appropriate optimizations for them. It can utilize specific features of certain processors (i.e. SIMD instruction sets such as SSE) and uses so called "lazy evaluation" to avoid performing unnecessary operations (Eigen, 2013). Additionally, all these optimizations are enabled automatically. A programmer does not have to select and enable them explicitly. The structure of the second implementation is similar to the first one. It obviously does not include Matrix nor Vector classes as these are provided by Eigen. It does not provide any operators for the same reason. It only includes the MMatrix convenience class, the Tangent class and its derivatives. It is not compatible with the custom Point2D class template, and points are represented as rows of a 2D matrix (this also ensures compatibility with the biomechanical toolkit). As a consequence computing the same interpolated value is slightly more complicated: result.row(i)=vecUTransposed*matM*points.block<4, d>(i, 0). As one can note, it requires row and block manipulation. This implementation will be referred to as "author's+eigen" in the reminder of this paper.

Numerical experiments

In order to compare the implementations mentioned above, a numerical experiment is designed. It consists of generating interpolated data (2D and 3D) using each implementation (the author's one and the one utilizing Eigen) of each algorithm (BP, CM12, CMP). The following pseudo-code demonstrates how the algorithms are tested.

```
for size:=4 to 480 step 4 do
    for steps:=5, 10, 20 and 100 do
        for test:=1 to 20 do
            data:=generateRandomData(size);
            startMeasuringTime();
            interpolatedData:=interpolation(data,steps);
            stopMeasuringTime();
        end for;
        time:=computeAverageTime();
    end for;
end for;
```

where: *size*– number of points used for interpolation, *steps* – number of points being generated for consecutive pairs of input points.

Tests for each combination of data type/implementation/algorithm are performed for varying data-set sizes and the number of steps in each interpolation range. Each test (for particular parameters) is repeated 20 times in order to obtain average time. The total of 114 240 test are run.

Results

Several interesting conclusions can be drawn based on the obtained results.

In the case of the author's implementation for 2D data, the CMP method is the slowest and the BP method is the fastest (Fig. 3). This is because the BP method doesn't require any additional computation of tangent vectors (they are "built in" in the M

matrix). On the other hand, CMP and CM12 need tangents to be computed. Tangent computation is the most complicated one for CMP. As can be seen in fig. A, the relative influence of tangent computation decreases as the number of steps in the interpolation range increases (tangents are computed only for the first and last point of this range). Results obtained for 3D data are similar.

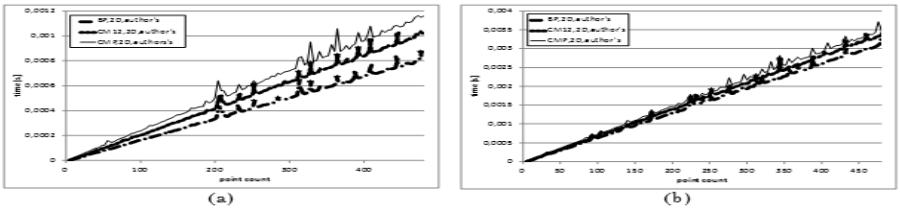

Figure 3. **Execution time for the author's BP, CM12 and CMP implementation for 2D data, (a) 5 steps, (b) 20 steps in interpolation range** *(author's development)*

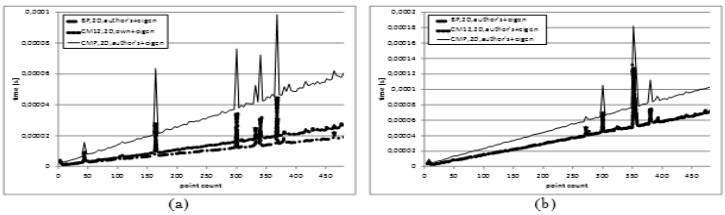

Figure 4. **Execution time for BP, CM12, CMP implementation utilizing Eigen for 2D data, (a) 5 steps, (b) 20 steps in interpolation range** *(author's development)*

Implementation utilizing Eigen in certain cases behaves differently. In case of 2D data and small number of steps (Fig. 4a), the CMP algorithm is the slowest and the BP algorithm is the fastest. Results obtained for CM12 lie in between. However, for 20 steps and above, differences between BP and CM12 diminish (Fig. 4b).

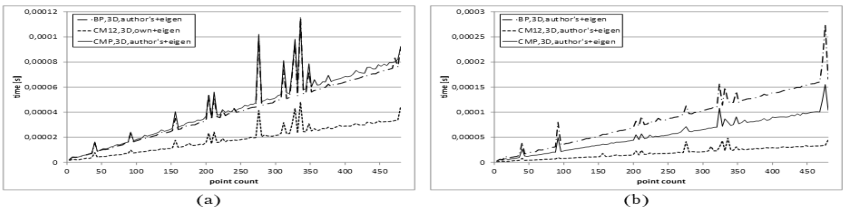

Figure 5. **Execution time for BP, CM12, CMP implementation utilizing Eigen for 3D data, (a) 5 steps, (b) 10 step in interpolation range** *(author's development)*

In the case of 3D data and 5 steps, the CMP method is the slowest, BP is second and CM12 the fastest (Fig. 5a). This changes when the number of steps is increased and for 10 steps: the BP method is the slowest and CMP is second (Fig. 5b). If the number of steps in the interpolation range increases further, the order of methods changes again and is as follows: BP, CM12, CMP (Figure 6b).

It is also worth noting that, based on the figures 3–6, the computation time depends linearly on the data-set size. An exception to this rule is 2D/3D implementation utilizing Eigen, when the implementation is requested to generate a large number of interpolated points (100 steps). As can be seen in Fig. 6, the computation time

sharply increases for certain number of points. This happens for all interpolation methods and for both 2D and 3D data. Interestingly, in all cases the increase appears after processing approximately 66000 point coordinates (for 2D data: 332 points * 2 coordinates * 100 steps, for 3D data: 220 points * 3 coordinates * 100 steps). This is probably caused by the fact that the Eigen library uses SIMD (single instruction multiple data) instructions available in modern computer processors. These instructions allow for performing the same operation (such as addition or multiplication) on several pairs of arguments. After crossing the above mentioned threshold, Eigen needs to increase non-linearly the number of SIMD instructions it uses.

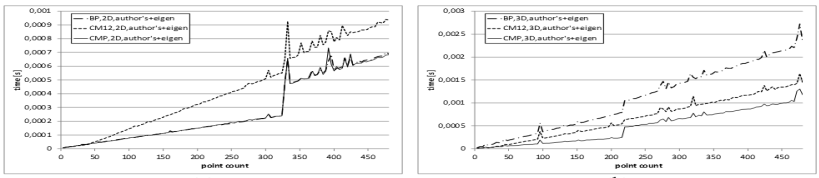

Figure 6. **Execution time for BP, CM12, CMP implementation utilizing Eigen, 100 steps in interpolation range (a) 2D data, (b) 3D data** *(author's development)*

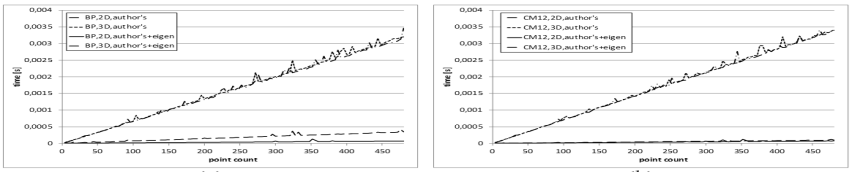

Figure 7. **Execution time for author's implementation and one utilizing Eigen for 2D and 3D data, 20 steps in interpolation range (a) BP, (b) CM12 algorithm** *(author's development)*

A conclusion that can be drawn based on Fig. 7 is that, in the case of the author's one, the implementation data type (2D/3D) does not matter. Execution time for 2D and 3D data are almost identical. This is caused by the super-scalar architecture of modern computer processors - multiple ALUs (arithmetic logic units) that allow for executing multiple instructions at the same time or by compiler optimizations that utilize SIMD instructions. The Eigen version is different in the fact that the BP method requires noticeably more time in the case of 3D data (Fig. 7a). Interestingly, other algorithms (CM12 and CMP) run in approximately the same time in the case of both 2D and 3D data (Fig. 7b).

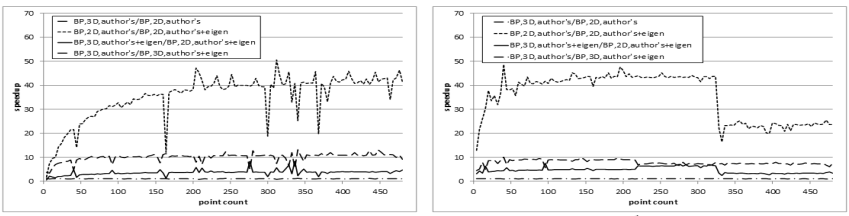

Figure 8. **Speedup achieved for BP algorithm, (a) 5 steps, (b) 100 steps in interpolation range** *(author's development)*

Figures 8 through 10 show speedups that were achieved by reimplementing interpolation methods with the use of the Eigen library. In the case of the BP method, the Eigen version is 40 times faster for 2D data and 9 times faster for 3D data (Fig. 8). Smaller speedups for low point counts are caused by overhead related to function calls. Speedup more quickly reaches "saturation level" if the number of steps in the interpolation range is higher (Fig. 8b).

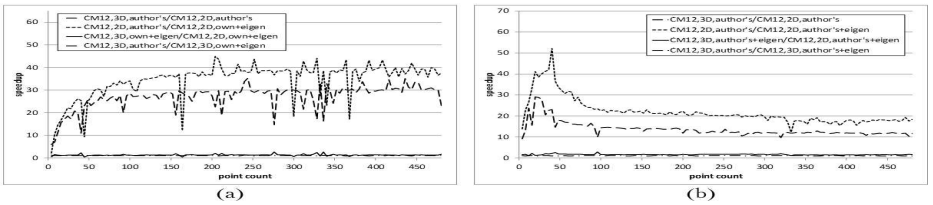

Figure 9. **Speedup achieved for CM12 algorithm, (a) 5 steps, (b) 100 steps in interpolation range** *(author's development)*

Speedups achieved for CM12 are similar, but they vary depending on the number of steps. For 2D data and 5 steps, the speedup is 40–45 times but only 20x for 100 steps. In the case of 3D data these figures are: 28 and 12 correspondingly.

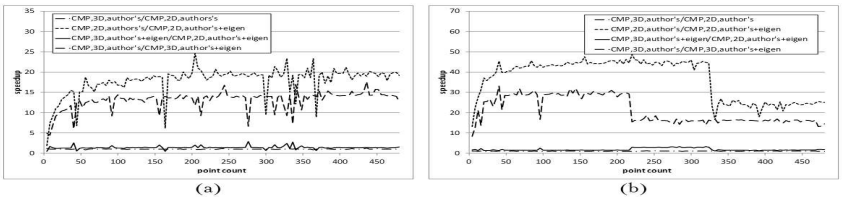

Figure 10. **Speedup achieved for CMP algorithm, (a) 5 steps, (b) 100 steps in interpolation range** *(author's development)*

Finally the speedups for the CMP algorithm range from 20 to 43 in the case of 2D data and 12 to 30 times for 3D data.

Conclusion

This article describes different implementations of the selected interpolation methods. They can be used for various purposes. But regardless of their application, their efficiency can be of crucial importance.

Three interpolation methods are implemented and compared: standard Catmull-Rom spline (CM12), modified Catmull-Rom Spline (CMP) and Blended Parabolas (BP). Two implementations are created ("author's" and "author's+eigen") in two variants (2D and 3D data). Computation time under different conditions is compared.

More conclusions are presented in the discussion above, but the two most important one are: (1) data dimensionality does not necessarily increase computation time (or the increase is relatively small), (2) using the Eigen library makes the implementation of interpolation algorithms more complicated, but it offers a significant increase in speed.

Preferences:

Biomechanical Toolkit (http://code.google.com/p/b-tk/).

Eckel, B. (2002). Thinking in C++, Helion.

Eigen (http://eigen.tuxfamily.org/index.php?title=Main_Page).

Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition), Addison-Wesley Professional.

Guennebaud, G. (2013). Eigen: A C++ Linear Algebra Library. Libraries for scientific computing. Ecole Polytechnique, May 15th.

Kopniak, P. (2010). Motion Capture Data Visualization Developed with the Utilization of jMonkeyEngine, Szczecin 01/2010; In book: Computer Graphics. Selected Issues, Publisher: University of Szczecin.

Kopniak, P. (2011). Budowa modeli 3D w edytorze Blender dla silnika gier jMonkeyEngine., Prace Instytutu Elektrotechniki, zeszyt 249, Warszawa.

Kopniak, P. (2012). Rejestracja ruchu za pomoca urzadzenia Microsoft Kinect. Pomiary Automatyka Kontrola, VOL. 58, Wydawnictwo PAK, Warszawa.

Parent, R. (2008). Computer Animation: Algorithms & Techniques, Elsevier.

Podeswa, H. (2009). UML For The IT Business Analyst (2nd edition), Cengage Learning PTR.

Tom's Hardware - How To: Building Your Own Render Farm (http://www.tomshardware.com/reviews/render-farm-node,2340.html#)

Стаття надійшла до редакції 17.07.2013.