Signal Convolution

Frank Hanisch
WSI/GRIS University of Tübingen, Germany
fhanisch@gris.uni-tuebingen.de

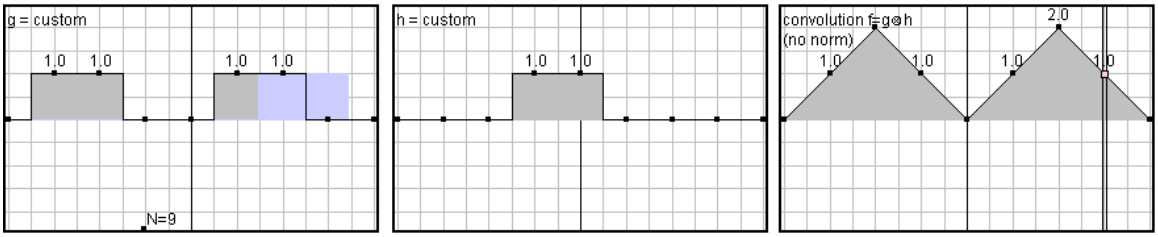

Figure 1: The teaching material convolves two signals (left/middle). Users can check the algorithm's correctness visually by analyzing the samples. Continuous and discrete views can be compared by changing the sample rate.

Abstract: *This interactive teaching gem shows the convolution concept for continuous signals and discrete samples. Signals occurring frequently in computer graphics education are predefined; others can be drawn or integrated easily. We explain how the material can be used for in-class demonstration and for homework, and describe the experiences we made with two courses on image processing and introductionary computer graphics. Student reading is provided.*

Keywords: Signal Processing, Filtering, Sampling, In-class Demonstration, Programming Homework

1 Introduction

Convolution is an integral concept of computer graphics (CG) curricula to explain, for example, aliasing, spline modeling, or image filtering. We have developed an interactive Java applet for demonstrating the core subject in-class and for assigning it as programming homework and student-side exploration. The idea stems from experiences we made with publicly available, interactive material, mostly Java applets. One of our own applets (Klein et al. 1998) already let learners convolve two unit boxes by direct manipulation. There, time was kept to a minimum for associating the convolution action with the concept. But the material was limited and could not be reused when the concept recurred later in class, for instance when sampling with a Dirac series, when interpolating with a triangle, in Gaussian smoothing, and in signal reconstruction. So we required additional, free material.

The Exploratories' convolution applet (Spalter and Simpson 2000) transfers a mechanical teaching material, layering and sliding of overheads. Users can draw two signals and convolve them physically by moving a slider. But signals are not pre-defined, which slows down work. Multiple convolutions, required for example to demonstrate spline basis generation, can hardly be performed in-class, as the convolved result must be copied to the input signals manually. The Joy of Convolution (Crutchfield and Rugh 1997) provides self-drawn signals and the convolution action, too, and includes pre-defined signals. Because these target audio processing, the demonstration still requires live drawing. Instructors may choose between applets for discrete and continuous signals. Included formula, parameter labels, and time-based system did not match our student reading, but we could bypass these obstacles in an oral demonstration.

However, we found the materials difficult to be applied for student-side exploration. The offered operations are simply not flexible enough to handle all the scenarios mentioned above. Consider reconstruction filtering with Blinn's (1989) NICE filter, which is constructed by multiplying two truncated sinc signals – sin(x)/x –, one of them stretched 1/3 horizontally; in the Fourier domain their dual signals are convolved. With the materials at hand, students cannot reproduce this process or explore similar filtering. Further, the materials miss source code, so learners can neither study implementation details, nor can the material be used as starting point for programming.

2 Educational Goals

Our goals in developing this teaching gem were to provide instructors with an in-class demo for the convolution concept as it appears in the core CG curriculum, and to enable learners exploring the concept in-depth. The source code should be usable for programming homework.

Learning objectives are as follows. First, learners should remember the convolution concept and use related vocabulary, which can be found, for example, in Glassner (1995). A novice should be able to write down the one-dimensional convolution $f = g \otimes h$ with signals g and h: $f(x) = \int g(a)\, h(x-a)\, da$. For self-studies he should know that authors may prefer other symbols and labels; x denotes a spatial system, whereas t would denote time. Knowledge includes commutative law and linearity, time-shifting the input causes the same shift in the output. Learners should remember that 2D convolution works accordingly.

Depending on the actual context, a CG student should understand convolution with examples from lecture or reading. Signals in CG represent images and functions; they are convolved with a kernel. Most kernels are symmetric with limited support; they filter the original signal and usually smooth it. Convolving with a box increases the signals' continuity by one. Two boxes convolve to a triangle, repeated box convolution converges to a Gaussian; the respective support length can be explained. In Fourier theory, convolution is dual to multiplication: F=GH, large letters denote Fourier transformed signals. The learner should know that the proof centers on $e^{-a} = e^{-x}\, e^{-(x-a)}$. The Dirac impulse copies the signal. Having understood linearity, the learner can now imagine effects of mixing in a third signal, and of scaling inputs in time or intensity.

Another objective is how convolution is applied in CG-related fields. In modeling, the B-spline is generated by convolving the control polygon, over and over again, with a box. Convolving a signal with a Dirac series produces copies, and their possible overlapping explains aliasing. A sample & hold display corresponds to convolving d-distant points with a d-sized box. A double-sized triangle would interpolate the points linearly. For reconstructing a signal, we would ideally convolve it with a sinc; in practice, with a NICE filter (see above).

At last, computer science students should be able to implement the convolution, mainly the infinite integral. According to the signals' internal representation and view resolution, the learner must decide on boundaries, sampling, and normalization. The learner should choose an appropriate signal and kernel, for example (0 1 1 0 0 1 1 0) and (1 1), convolve them programmatically, and verify the result (0 1 2 1 0 1 2 1) manually. An exemplary output with our material is given in Figure 1.

3 Methodology

The authors of Joy of Convolution show discrete-time applets in class and leave the continuous-time applets entirely to self-studies. The transition from discrete to continuous view is not explained. Our approach proceeds reversely: we introduce the continuous case, explain the transition in class, and let students construct the discrete algorithm. Students are given a ready programming skeleton and working examples with the above material. Learning occurs in small groups: for a few days, students discuss the concept, share ideas, and develop the algorithm. Forum and wiki support inter-group discussion.

Our developed teaching gem supports the given learning objectives with the following unit tasks, operational building blocks with minimal interdependencies that keep the material from getting too application-specific:

1. Set input k to a box
2. Set input k to a sinc
3. Set input k to an impulse series
4. Set parts of input k to zero
5. Draw parts of input k
6. Set input k to the convolution $g \otimes h$
7. Set input k to the multiplication $g \bullet h$
8. Scale input k by α: $k \rightarrow \alpha k$
9. Shift input k by s: $k(t) \rightarrow k(t-s)$
10. Stretch input k by α: $k(t) \rightarrow k(\alpha t)$
11. Modify sampling rate
12. Normalize output

Out of these unit tasks, the instructor can compile his in-class demo. The first three tasks include common signals in CG. Task 4 enables users limiting the support, task 5 represents custom signals. Tasks 6/7 repeat convolution and dual multiplication. Tasks 8-10 derive from linearity and time-shifting properties. Note that impulse, triangle, Gaussian, and NICE can be constructed. We specify the latter according to Blinn (1989) with the following task series (letters denote target input signals g and h):

2g - 1h – 7g - 2h - 10h - 7g - 10g

In other words, unit tasks enable us to formally define an interaction. For simplicity, we left out concrete function parameters or the scaling/shifting values α and s, but they could be added similarly to the signals' letters. The above task series can be read as follows: "Multiply (7g) the ideal sinc filter (2g) with a truncation box (1h), then multiply it (7g) with a stretched sinc (Lanczos window, 2h-10h), and stretch it (10g)".

The dual task in the Fourier domain would use the dual operations: "Convolve (6g) the ideal box (1g) with a sinc stretching (2h), then convolve (average, 6g) it with a stretched box (1h-10h) and shrink the result (10g)" can be executed as task series:

1g - 2h - 6g - 1h - 10h - 6g - 10g

The application of this task series is shown in Figure 2. Demonstrating aliasing would comprise the task series 5g - 4g - 3h - 10h - 10h: "Draw signal g, limit its support, set kernel h to a Dirac series, and vary its stretching". Other in-class demonstrations can be specified in the same way.

Offering tasks like stretching supports exploration: stretching a box towards the constant (DC) blurs the output towards the average, shrinking it again towards the impulse resharpens the output. Stretching can also help in discovering that support lengths sum up. We have excluded some signals (triangle, Gaussian, NICE) from selection to force the user construct them by convolution. During scaling, the signal maximum snaps to multiples of 0.5 and to the value for which the kernel becomes normalized.

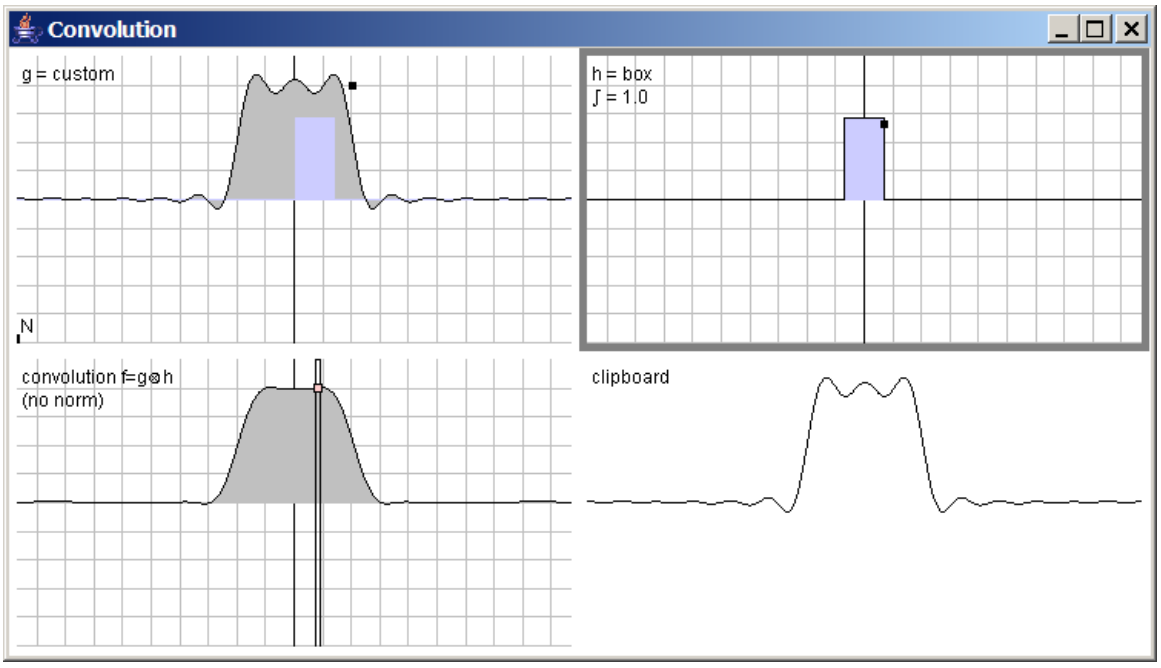

Figure 2: Here, we explore Blinn's (1989) NICE filter in frequency domain. We have convolved the ideal box with a sinc stretching and copied the result to the clipboard. Now we get rid of the ripples by averaging it with a one-cycle wide box. Signals can be drawn, scaled, and stretched by the rectangular handles. That way we magnified the input and normalized the box integral; alternatively, we could use the built-in normalization.

Besides visualizing the integral's calculation at a given location x, we sample both inputs g_s and h_s and visualize the convolved samples $f_s = g_s \otimes h_s$. With that, the user can verify results manually and explore the transition between continuous and discrete view by varying the sampling rate (task 12; see Figure 1). When N is near the screen resolution, we hide the samples and switch back to the continuous view. In our assessments, we have found that task 12 is the most difficult to comprehend. The following instructions have helped our students: The user should start with sampling at the grid resolution by dragging the "N" handle to the first grid point left to the center at x=-0.5. This corresponds to N=9 sampling points. Interaction is easier when the application window is maximized. It may further help to interpret the rectangular sample points as pixel values: each pixel spans exactly two grid cells, one left and the other one right of its rectangular sample point. To perform the discrete convolution (0 1 1 0 0 1 1 0) ⊗ (1 1) mentioned above, the user now draws the pixel blocks seen in Figure 1. Each 0 corresponds to two zero cells, each 1 to two unit cells. We have simplified drawing by snapping to partially linear signals.

Users can operate all unit tasks by direct manipulation. Figure 3 overlays the graphical user interface with the actual mouse and keyboard interactions. A quick reference is included in the accompanied HTML/applet page. The application can also be executed as platform-independent, executable Java archive (JAR) or as Windows executable (EXE). Users are required to install Java 2.

The included makefile enables users to compile the application. Our teaching gem includes full source code, except for the convolution algorithm that is assessed as programming exercise. The Java class applets.convolution.Signals, contained in the accompanied source package, presents a short, 50-line code module that demonstrates how two signals can be multiplied, packed in the body of the method multiply(double[] signal, double[] kernel, double[] result, int norm). Educators can assign students the second, empty method convolve(double[] signal, double[] kernel, double[] result, double delta, int norm). The implementation of the convolution algorithm is straightforward and students can compare their coding with the multiplication code, but complexities arise when they start thinking about convolving at the signals' borders, or when they consider normalization. Students will recognize that, in the discrete case, this involves a delta weighting for each sample point. Note that after starting the makefile, the HTML/applet will contain all student modifications, while the JAR/EXE versions still contain the original compiled code; if the convolution code is left empty, the convolution result will be zero.

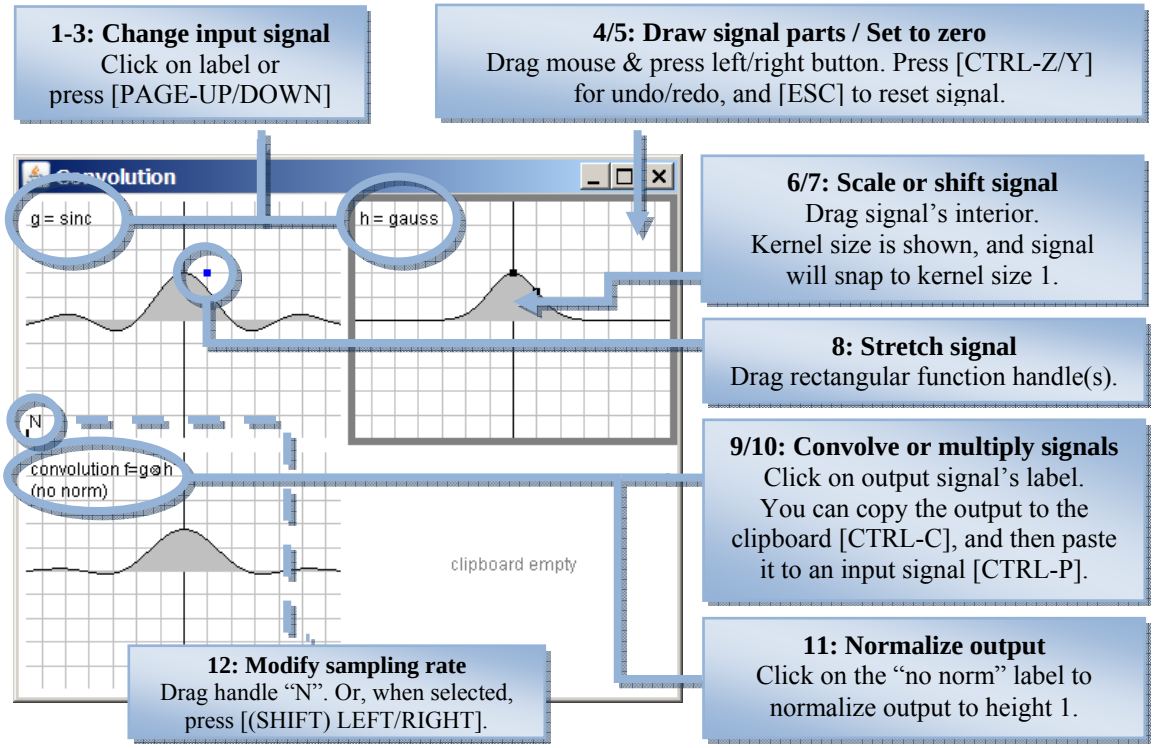

Figure 3: Annotated user interface of the teaching gem. Numbers correspond to unit tasks described in the text.

We have based our programming architecture on a Java graphics engine with views and controllers in which we render a list of graphics items (package ilo.awt). The controllers perform the model-to-view transformation, selection and dragging; they further encapsulate undo and clipboard functionality, as well as shape recognition for partially linear signals. Signals are represented as floating point arrays and they are connected to a text parser for custom input and draggable function handles (package ilo.math). The applet package contains the class applets.convolution.CustomFunction, which showcases how users can integrate custom functions with a few lines of code – together with the rectangular interaction handles.

The applet itself holds a view/controller pair for varying the sampling and selecting pre-defined functions (package applets.convolution). Objects are available to the scripting interface, so users can customize the applet, for example to enter other signals like a high pass or a ramp. The engine queries CSS information before rendering its graphics items; most visual parameters can be styled textually. Examples would be to hide the samples or to highlight the copying effect of the Dirac series instead of the integral's calculation. A default color scheme for primary and secondary foreground, background, filling, and emphasis is defined in the class ilo.awt.GraphicsStyle.

4 Assessment

We have collected the stated learning objectives for convolution in 2005 with two core courses on Image Processing and Introductionary Computer Graphics, where students had to explain and reduce aliasing effects in texturing respectively Moiré effects in images. Both courses used the same material and inquiry for the convolution concept. Individual data was collected from 60 of the 63 students; three were excluded from the study as they missed the final deadline. Their field of study was either Computer Science (43) or Bioinformatics (17), gender ratio was 82% male to 18% female. The average age was 23 years. A pre-test assured that student previous knowledge did not differ significantly.

Students worked in triads. They had to implement the sampling algorithm with different filtering strategies. In an oral exam, they had to demonstrate their application, to explain the effects of their code, and to reproduce Fourier and sampling theory, including the convolution concept. They had 12 days to finish their programming and to prepare for the oral exam. With respect to the convolution concept, the analysis of our oral examinations showed these major lacks in understanding and application:

- Commutative law and linearity could be stated but not applied
- Signals and convolution could be drawn but often with a false frequency
- Scaling for concrete applications was generally not understood
- Normalization was generally confused

The results let to the new material at hand, which was assessed in 2006 in the same courses with the same assignments. The interactivity was given to the students before the exercise was handed out. After the oral examination, we inquired 12 individual students and let them demonstrate the convolution concept directly with the teaching gem. All of them stated they had used it to gain an understanding of the concept. We tested commutative law and linearity by letting them add, scale, and shift signals (tasks 5, 8, 9, 10) and letting them guess how the convolved result would look like. Because our student reading shows mere box convolution with two unit boxes that results in an already normalized triangle with unit area and unit height, we questioned how two double-sized boxes would convolve. Then we entered discussion on possible approaches to normalization (by height, by area) and applications in image processing. Depending on the individual outcome, we explained other, more advanced applications.

All students could handle and explain the various signals and their convolutions intuitively – commutative law and linearity could generally be demonstrated with the applet. Four students stated they had explored normalization, – and only two of them could explain scaling correctly, not only for double-sized boxes but for arbitrary signals. The fact that signals are not normalized by default (task 12) makes us think the remaining users have not investigated convolution with non-uniform boxes. However, we have no objective data on that. Based on this feedback, we see it crucial to instruct students that their reading might differ in scaling and normalization; some authors are not too explicit on that, and even Blinn's clear statement in his introduction to the NICE filter (1989) could be overlooked.

Furthermore, we confronted students with an additional, false normalization that violates commutative law, as we were curious about their analysis skills. None of them discovered this "bug". Teaching material

might be considered true by students, so we rather suggest training analysis skills explicitly, for example with multiple-choice tests or similar setups (Hanisch et al. 2005).

5 Conclusions

This teaching gem has enabled us to effectively demonstrate the convolution concept in CG classes and to assign it for student exploration. The provided unit tasks cover learning objectives from comprehension to application to programming. While we have given formal task series for teaching filter design (Blinn's NICE filter) and aliasing, many others are possible, and we suggest CG educators develop and contribute further task series, for example by adding their task series to the CGEMS repository.

References

Blinn, J. 1989. Return of the Jaggy. In IEEE Computer Graphics and Applications 9(2), pp. 82-89.

Crutchfield, S. G. and Rugh, W. J. 1998. Interactive Learning for Signals, Systems, and Control. IEEE Control Systems Magazine 18(4), pp. 88-91. http://www.jhu.edu/signals

Foley, J. D., van Dam, A., Feiner, F., and Hughes, J. F. 1990. Computer Graphics, Principles and Practice, Second Edition, Addison-Wesley, Reading, Massachusetts, ch. 14.10.3, pp. 629-633.

Glassner, A. S. 1995. Principles of Digital Image Synthesis 1, Morgan Kaufmann, San Francisco, ch. 4.5, pp. 155-164.

Hanisch, F., Görke, J., and Straßer, W., 2005. The educational use of live graphics gems: a walkthrough for aliasing, in: Technical Report WSI-2005-9, WSI/GRIS University of Tübingen.

Klein, R., Hanisch, F., and Strasser, W. 1998. Web based Teaching of Computer Graphics: Concepts and Realization of an Interactive Online Course. SIGGRAPH 98 Conference Proceedings, Addison Wesley. http://www.gris.uni-tuebingen.de/projects/grdev

Spalter, A. M., and Simpson, R. M. 2000. Integrating Interactive Computer-Based Learning Experiences Into Established Curricula. Proc. of ACM ITICSE 5, pp. 116-119. http://www.cs.brown.edu/exploratories