AC 2007-1733: LEARNING ABSTRACT INFORMATION THEORY ON VISUAL DATA: AN INTEGRATED COURSE ON WAVELET-BASED IMAGE COMPRESSION

Thomas Richter, Technische Universitat Berlin

Sven Grottke, Technische Universitat Berlin

Learning Abstract Information Theory on Visual Data: An Integrated Course on Wavelet-Based Image Compression

Abstract

We describe the implementation of and our experiences with a capstone course on wavelet based image compression held at the University of Technology Berlin in the years 2002 to 2006. This course has been designed as an “integrated project”, which means that it combines lectures, seminar talks to be prepared and held by the students, and a programming part.

The design goal of this course has been to provide all the necessary theoretical knowledge to understand the concepts behind image compression technologies, such as JPEG2000. We are also aiming at simulating the work-flow as found within an IT company as realistically as possible, preparing electrical engineers and computer scientists as well as possible for their professional life. This training does not only include the technical, but also the social skills required to successfully complete larger projects.

The subject of image compression offers the advantage of requiring a solid knowledge on terms of information science such as entropy, distortion, quantization, Fourier and wavelet-transformation, but also offering a direct visual feedback of how these techniques perform. Therefore, we believe that image compression is an attractive topic to be used for a capstone course.

Traditionally, a course would assign weekly programming exercises to the students; however, we believe this to be unsuitable for a capstone course as it does not simulate the work-flow of a professional software development team; furthermore, it does not require the degree of team-work we deem critical to modern software development. Thus, we divide students into groups of two to four people and assign each team to one sub-task of an image codec and provide some boiler-plate code of our own. Much to their astonishment, students soon find themselves spending a considerable amount of time with project management and coordination activities. That means, teams have to design interfaces and data structures to combine their efforts to create a working project, which adds an often underestimated social component to the course. With some guidance from the teachers, students have always been able to supply a working code at the end of the semester. Needless to say, the thrill of having a nontrivial working program at the end of the course is a major source of motivation for our students and adds much to the satisfaction and positive feedback we receive.

1. Background

The field of image compression often attracts students of various fields: the technology is well-recognized in today's world, let it be as images found on the Internet, or as the compression engine in digital cameras. Students of technical fields are aware of the technology, and equipped with their basic knowledge of information technology, soon become curious about how these things work. Furthermore, compressing images also means **looking at** images – it is visual technology with a mathematical background.

Of course, things were not much different for the authors of this paper, even though we came from almost completely unrelated fields: Mathematical physics and computer science. In 2000, the IT bubble gave us a chance to explore our knowledge in a tiny startup company whose name is likely unknown to anyone else, and the breakdown of the bubble to learn the not so bright sides of it. We thus moved back from industry to university, hoping to pass some of our experience to students.

Our first try, a regular lecture on JPEG2000[2] – about the only product our company ever sold – was not really a success. The matter **is** complex, and students often find that understanding the subject requires more mathematics than they would like, and presenting something as visual as image compression on a blackboard, and maybe some images here and there, did not really help to improve matters. Hence, the course had to be redesigned completely.

2. The Design of a New Course

We started with what we expect our students to learn: Of course they should know how JPEG[1] and its related standards work. But we believe that just knowing and managing the **technology** is not sufficient to prepare them for their further careers. One of the skills one needs in industry is the ability to plan and manage large projects. And the team has to be organized as well: Students **have to learn** how to cooperate with their colleagues, an often surprisingly complex task in a large team, and a skill usually not taught at university level. Last but not least, as we are working at the institute of mathematics of the TU Berlin, we also considered it important to make our students understand the scientific foundations of the field.

Our idea was to simulate the work-flow of industrial software development as close as a university course would permit. The goal of the course is to provide a fully working image compression program to a "virtual customer" at the end of the semester – that is what we tell our students – and learn all the skills required for this as the progress; the latter often means more than our students would initially believe. For that, after an initial introductory lecture, we break our audience into groups of two to four students and let them pick one of the sub-projects. Each sub-project corresponds to a key component of a realistic image codec. This means that each student group gets **a different** assignment; this is realistic, but not without problems, as we will show in section 4.

To prepare students for their task and to provide the required foundations, we make them give a one hour presentation to the whole class. In their presentation, they explain the technological backgrounds of their assignment, as well as their basic work plan for the rest of the term. The necessary material, mostly technical and scientific articles, are provided by us during the initial lecture. This approach might seem a bit harsh, but it is part of our philosophy: it is just realistic that a professional IT engineer has to implement a project from the papers, and must learn the background by himself. Needless to say, we offer all the help to the students we can; in addition to keeping our offices open for them whenever they need assistance, students are required to talk to the organizer of the course at least one week in advance to their talk to get it straight, clear up the misconceptions and fix any errors. Thus, we do have some quality control for their talks.

Despite some boilerplate code provided by the organizers, the students not only implement, but also design their code themselves. As we find year after year, especially the latter causes major headache for our students. Apparently, up to this course, students never spend much thought in how to write a larger software project. It is also hard to make them communicate with each other, and not with the administrator to get the interfaces of their software modules straightened out. For more experiences on this, see also section 4.
At the end of the term – six months at our universities – the image codec written by the students has to pass a formal acceptance test, similar to what is required for professional projects. We keep testing the progress of the project within the semester and provide feedback and help, such that up to now every class had a working and acceptable "product" at the end of the semester.

3. Organization of the Course

We initially designed the course[10] around five modules, as found in a wavelet based image codec: Color transformation, wavelet transformation, quantization, rate allocation and entropy encoding.

To give a brief overview: the color transformation computes luminance and color coordinates from the red, green and blue intensities of the image, very much like what happens in color TV. This step provides a first decorrelation of the color planes of the image. The following wavelet transformer[6,7,8] (DWT in Fig. 1) decorrelates the original image by transforming it from the spatial into a frequency/scale domain, very much like the Discrete Cosine Transform in JPEG-1 does[1]. It is by itself not lossy, except for numerical round-off errors. Data reduction and thus compression is the task of the following quantization step, which is controlled by the rate-allocator. The purpose of the latter is to provide quantization parameters such that the image quality becomes maximal for the user provided output file size. Context modeling detects edges and corners in the image and provides the following step, the entropy coder, with a classification of the quantized coefficients. The entropy coder finally removes redundancy in the data by classical algorithms like Huffman coding[4] or arithmetic coding[5]; see Fig. 1.

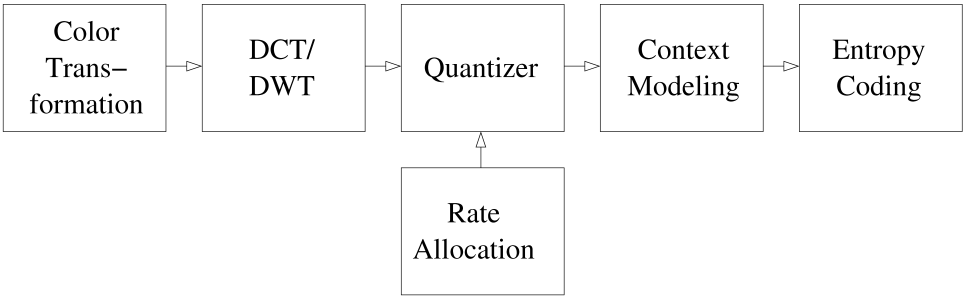

Figure 1: Layout of an image compressor, arrows indicate the data flow (see text above for details)

We also provide a fifth backup project that uses an alternative rate allocation scheme, the so-called zero-tree coder by Shapiro et al[3]. We later dropped the quantization module in favor of a

context modeling module for reasons discussed in section 4 and provide the quantization software and talk ourselves.

Some boilerplate code, namely to read and write images in the uncompressed PNM file format and service classes, including the color transformer and the quantizer, are also provided by the organizers, along with the architecture that builds the code. We picked C++ as the language of our choice as it has some relevance to the market, especially in still image and video compression. Even though Java gains a lot of popularity, it is rarely picked for computationally intensive tasks in the industry. The operating system of our choice is GNU/Linux, as provided by the computer lab of our department.

The course itself[10] is scheduled to take up four hours per week, two hours spend in a lecture hall, and two hours in the computer lab where students have to show up to work on their code. Even though students could in principle write their software at home, we require them to meet in the lab to be available for their colleagues for discussion and help. Of course, the lab remains available to students outside theses hours as well.

The schedule of the course is rather delicate: Some modules depend on the result of others and have to be finished early, while others may delay their work to the mid of the semester. For example, pretty much everybody requires the output of the wavelet transformation step, but the rate allocation could be completely omitted, at least in theory. To address these dependencies, we fix the dates for the student talks in advance and set deadlines for the projects. Some students have to volunteer for earlier projects, e.g. the wavelet group has to complete early, while others need to complete their module rather quickly at the end of the semester, e.g. the rate allocator group.

To allow the early groups to read into their topic and get their talk prepared, the first three lectures on color spaces and color transformation, on the project architecture, and on the revision control system that maintains the project sources, are given by us. The middle part of the semester, usually up to the Christmas break, is spent with student talks. During the remaining term, we discuss extensions and refinements of the technology, while students are busy implementing their modules. Subjects such as the JPEG 2000 EBCOT rate allocator[2], which we do not require to be programmed by students due to its complexity, and mathematical properties of wavelets are discussed during this time. Since all the foundations are now available, we are also able to provide a deeper insight into the mathematical backgrounds: DWT and DCT are an ideal motivation to discuss function spaces and Fourier transformation, whereas context modeling and entropy coding are applications of probability theory and random Markov fields. Depending on available time and request, alternative compression schemes, e.g. JPEG-LS[9] or PNG are also discussed here. The final week is spent on preparation for the acceptance test and we often found it necessary to extend the course by one week for additional bug fixing.

Passing the course finally requires not only the talk and the delivery of working code, but an oral exam to be taken by the students some time after the course. The grade is a weighted average of the grades of the talk, the quality of their software and the grade of the final exam. We initially allowed the oral exams to be taken in groups, but switched over to individual exams for reasons discussed in the next section.

4. Lessons Learned

One very general problem of the course is to setup the available projects such that their difficulty level is comparable. Specifically, in the very first course we observed that weaker students were attracted to the quantization project, because it is simple as far as the implementation is concerned, even though the mathematics are quite complex. However, the latter is often not realized by students, and causes unpleasant surprises when the talk has to be prepared. We thus decided to make the quantization talk a regular talk of the lecture and provided a replacement project on context modeling in the next and all following years. To our surprise, it is much less popular than our backup project, the Shapiro-Zero-Tree coder, even though the latter requires even higher skills in software engineering. As a result, even weaker students are now regularly attracted by the more complex issues such as rate allocation and to our satisfaction they always managed to complete them successfully.

Teamwork and project management often cause more trouble than students initially expect. In particular, students frequently approach us, the organizers, with questions about the code or the interface definition of other teams. It is part of the learning process – ours as well as that of the students – to directly refer them to the corresponding team and to meet in a larger group if issues have to be worked out that require an even broader audience. We do not feel that the need of holding meetings is a bad thing – it will certainly become an important part of their professional life.

Related to this issue, software design -as opposed to just implementing code- also seems to be very unattractive to students. It is clearly much more convenient to accept a design as given and hack in the code than rather to think of the project as a whole, but we also need our class to understand the big picture. Designing the code is now done in one regular lecture and several meetings over the course of the semester, where the organizers act as “consultants” explaining the benefits and drawbacks of the ideas our students come up with. It happened once that one dominant student proposed a – in our view unattractive – design idea how to organize the wavelet bands, but we decided to let the group go. It turned out later on, unsurprisingly, that the proposed solution caused quite some headache. Nevertheless the course was completed successfully in time and no damage was done to the project, but students surely learned the importance of design decisions, and hopefully that the loudest persons are not always the wisest. Clearly, not intervening is always a risk to the course, but providing a working design in first place contradicts the learning goal. We thus restricted our role to moderating the design meetings and not making any final decisions. Needless to say, this often seems to surprise students.

We initially allowed students to take the oral exam in groups of three, but soon found this unsuitable to provide grades to individual students because there is often a single dominant student in the project who will answer most of the questions, making it hard to address the quieter members. Exams are now required to be taken individually.

It proved very helpful, though, to meet the students in advance to their presentations to discuss the talk they were about to give. This often clears out misconceptions, and gets the talk on focus

of what we think is important to learn and what is important for the mathematics that follows later in the semester. In addition, we ask the class to collect questions to the presenters while the presentation is running and add a couple of questions of our own. Initially, our classes considered our questions more of an inconvenience for the presenters. We now make it very clear to them that these questions are meant as a way of assisting them with their preparations for the exams, as they are a tool to point out what we consider relevant in the course.

In addition, we also give away the secret of the first question given to all participants: “Please draw a block diagram of an image codec”, i.e. reproduce Fig. 1, and explain the purpose of the boxes. It does not really take much time, and breaks the ice by allowing the examinee to get more comfortable in a stressful situation.

Concerning the organization of the course, the proper timing has been found very critical, as already mentioned in section 3. Students often underestimate the time to get the code running correctly, and only start working on the computers when it is already too late – unfortunately, this matches our own experience in the industry only too good. We will address this point in the next year by providing them with two software projects to be solved: One small introductory assignment that requires students to get familiar with their topic and the software architecture of the framework, and the actual code for the image compression codec. The former does not require interaction between the teams and thus does not cause any additional time constraints in the setup of the course, and we can still cancel it if it takes too much time.

The first project should ideally provide some useful data for the final, second code: For example, as an add-on project for the entropy coder module, we plan the assignment to write a program to measure the image statistics and compute the zero order entropy from that. We expect this to help students to get familiar with both the theoretical background required for their talk, and with the development environment by forcing them to work on the computer. We have yet to see how this idea will work out.

One of the ideas our students themselves came up with is to make them hand in a routing card for the exam; this forces students to come into the lab frequently, at least to get the card signed for every appointment we make, and thus requires them to work more regularly on the program.

5. Evaluation

The German university system does not require evaluation of courses, but we nevertheless asked our classes to fill out evaluation sheets provided by the student union to all lecturers on a voluntary basis. This evaluation sheet grades lectures on a range from 1 to 6, following the German school system, where 1 is the best and 6 the worst possible degree, very much unlike the US system. The evaluation sheets are collected anonymously at the end of the semester. In addition, we keep asking our students for comments and suggestions after the completion of the exam. At this point, the grade is already set and students can talk to the organizers more openly.

Our course consistently collects grades between 1.2 and 1.5 on average and makes this one of the highest graded courses of the institute of mathematics. We collected a lot of positive feedback, but a couple of critical comments as well. To quote one of the students: “This was the first

reasonable course I ever took" - it seems something must be wrong with the undergraduate courses in electrical engineering. It is often well appreciated that our course combines the solution of a practical problem, and thus a hands-on approach to the problem, while it also provides the theoretical background material. Thus, students seem to like learning how things like JPEG work by actually creating a working implementation of their own.

From the critical voices we collected, the two issues coming up most are that the high amount of mathematics in the course is disliked, and that the software design meetings have been found to be annoying. The first point is not negotiable – image compression is applied mathematics, and we don't believe that cutting the mathematics part down will improve the understanding. We try our best to make the mathematics visible – also in a very literal meaning – and found that this works much better than by just working on the blackboard.

Concerning software design, compromises have to be made. On the one hand, it is clearly easier for the students just to follow a fixed course set by the lecture. On the other, if engineers do not learn how to design software while they are in university, they have to in their jobs – which might cause them some serious trouble. We try to address this issue by acting as senior engineers in the design meetings.

Our experience with students is that we found the overwhelming majority of them very engaged and actively contributing to the project. A minority, often two or three individuals, take the course because they don't seem to have found anything more fitting. This relation is also mirrored in the final grades students collect. Unlike a Gauss-shaped plot one might expect, we are often proud to provide many very good grades, but also a couple of bad ones. Passing the course with a mediocre ranking seems to be hard – if you're interested in the technology, the course seems to be exactly what you need and it can pretty tough otherwise.

6. Future Work

One of the problems we definitely need to address is to make our students aware of the critical timing of the project, and that developing working software costs more time than one believes. We have yet to see how well our approach of assigning initial mini-projects to students work in this respect. A second challenge is to keep students convinced that software design is a serious issue. There does not seem to be a perfect solution to this problem, but anything we can do in practical programming projects seems to address this much better than a theoretical lecture on the topic.

7. Acknowledgments

The authors want to thank Prof. Sikora and the Institute for Nachrichtenübertragung for the kind cooperation and making this project possible. We further want to thank the Institute for Mathematics of the TU Berlin for financing and support. Th. Richter wants to thank Pegasus Imaging for sponsoring.

Bibliography

1. **William B. Pennebaker, Joan L. Mitchell:**
   JPEG Still Image Data Compression Standard
   Van Nostrand Reinhold, New York, ISBN 0-442-01272-1
2. **International Organization for Standardization:**
   JPEG2000 Part I Final Draft International Standard
   ISO/IEC JTC 1/SC 29/WG 1 (ITU-T SG8)
3. **Jerome M. Shapiro:**
   Embedded Image Coding Using Zerotrees of Wavelet Coefficients
   IEEE Transactions on Signal Processing, Vol. 41, No. 12, December 1993
4. **David A. Huffman:**
   A Method for the Construction of Minimum Redundancy Codes
   Proceedings of the I.R.E., September 1952
5. **Alistair Moffat, Radford M. Neal, Ian H. Witten:**
   Arithmetic Coding Revisited
   ACM Transactions in Information Systems, Vol. 16, No. 3, Juli 1998
6. **W. Sweldens:**
   Wavelets and the lifting scheme: A 5 minute tour
   Z. Angew. Math. Mech., 76 (Suppl. 2), 1996
7. **W. Sweldens,P. Schröder:**
   Building your own wavelets at home
   Wavelets in Computer Graphics. ACM SIGGRAPH Course notes, 1996
8. **W. Sweldens:**
   The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions
   Wavelet Applications in Signal and Image Processing III, Proc. SPIE 2569, 1995
9. **International Organization for Standardization:**
   JPEG-LS FCD-14495 public draft
   ISO/IEC JTC 1/SC 29/WG 1 (ITU-T SG8)
10. **IMCO Project Home Page:**Prof. Dr.-Ing. Thomas Sikora, Dr. Thomas Richter, Dipl.-Inform. Sven Grottke Dipl.-Phys. Marc Wilke at www.math.tu-berlin.de/~thor/imco