Computer Graphics Curriculum: a Programming Approach

D. Sobczyk & M.-S. Touzeau & J.-J. Bourdin

Laboratoire d'Informatique Avancée de Saint-Denis
Université Paris 8, France
{dom,mst,jj}@ai.univ-paris8.fr

Abstract

At University Paris 8 the computer science curriculum is focused on programming as a good technique to improve the skills of students and to improve the success of studies. It has been reinforced when France adopted the Bologna requirements. This approach is well adapted to our computer graphics courses. The results of these courses are discussed.

Categories and Subject Descriptors (according to ACM CCS): I.3.0 [Computer Graphics]: K.3.2 [Computer and Information Science Education]:

1. Introduction

As most computer graphics curriculums are developed for students of various backgrounds (major or non-major, computer science degree or computer and arts degree...), programming is no more the key of teaching computer graphics. This paper will present an example of the use of intensive programming for teaching computer graphics.

2. Background

After many workshops on computer graphics education, Coimbra in1999 [Cun99], Bristol in 2002 [Cun02], Hangzhou in 2004 [CHLS04] and Vienna in 2006 [BCFH06], the definition of a common computer graphics curriculum remains a goal to be achieved. In Vienna [BCFH06] the question of when to start teaching GPU programming was raised. With good reason, some of the attendees wanted this teaching to begin after the beginning course while others with as good reasons thought that as really too early. More generally the question of the part of programming in our curriculum has to be handled carefully. It is not only the question of which programming language to teach but also the simple idea of programming as a learning tool.

We will not pretend to circumvent the whole of this subject but only to give the first ideas to launch a debate on these questions.

3. A programming curriculum

The computer science curriculum of our university is oriented on programming. This choice has been made a long time ago and confirmed when we had to adapt our curriculum accordingly to the Bologna requirements. The reader may refer to Fuller et al. paper [FPA*06], to get an insight of the consequences of the requirements, but we can summarize the main objectives as:

1. the adoption of a common framework of readable and comparable degrees, (also through the implementation of the Diploma Supplement);
2. the introduction of undergraduate and postgraduate levels in all countries, with first degrees no shorter than 3 years and relevant to the labour market;
3. ECTS-compatible credit systems also covering lifelong learning activities;
4. a European dimension in quality assurance, with comparable criteria and methods;
5. the elimination of remaining obstacles to the free mobility of students (as well as trainees and graduates) and teachers (as well as researchers and higher education/administrators).

The second point made a great difference in France as we had previously a two-years long "DEUG" and a one-year long "licence". It was not possible to put them together and present it as the new undergraduate curriculum. We had to

adapt also to the labour market and it is not that easy with a three years long curriculum. Our bachelor degree is three years long and therefore the master degree that follows is two years long.

The first year includes four courses on programming:

- *Introduction to programming languages*. It seems to be a very broad presentation of the languages but in fact it's mostly the place where students have to learn how to write, compile, execute and correct a program. The current programming environment is the old `emacs`, a local `Lisp` is presented and `gcc` is used. Such languages as `pyhton` and `prolog` are also presented. Even for very simple functions our students have to spend a lot of time at debugging process. The main reason is that in early classes they have not learnt to write (even in French) without mistake.
- *Methodology of functional programming*. In this class students lear how to program with `Lisp`.
- *Imperative programming*, meaning a `C` programming class.
- *Logical programming* or *object oriented programming*.

To validate these courses students have to present a project. The projects are expected to take from ten to twenty work hours. Usually most of the students work more than twenty hours and some results are impresive. When they have worked that much students seem to have taken the habit of spending most of their time in front of computers trying to make better and better programs in order to impress their teachers or, at least, their fellow students.

The teachings of such matters as algorithmic or data structures begins after this first year. Therefore almost every algorithm presented in class has to be tested by the students. In second year a "project based learning" course has been introduced [MGJ06]. It is mostly a course on working as a team and designing a project, step by step, from scratch to the writing of the users' manual.

In our curriculum the computer graphics courses are given during the third year. A student may take up to three courses: "graphics programming", "image synthesis" and "GPU programming". Each of them represent 40 hours of class plus a large amount of personal work. The first is a prerequisite for the third. In our bachelor (three years) degree, none of these courses can be mandatory. But the first, at least, is very popular among students. It is included in a module with limited choice, each student has to take three of the four courses of the module. In reality, most years, only three of these courses are effectively on schedule and the graphics programming course becomes mandatory.

The students express a strong interest in these courses even if they are not compulsory. For example in 2006-2007, on 50 students of third year, all of them took graphics programming course due to the lack of choice, 15 of them (30%) took the image synthesis course and 18 of them (36%) took the GPU programming course. Most of them expressed interest to carry on in a master with strong computer graphics component and at least one of the majors left our university to join a more computer graphics master degree.

3.1. Contents of the courses

3.1.1. Graphics programming

At the end of this course the student should be able to use a modern graphics API to create a graphics application that can be integrated with other computer applications. In recent years the API presented has been OpenGL. In the course these points are studied:

1. OpenGL Machine
2. Geometric drawing
3. Visualization
4. Visualization and colors
5. Lighting
6. Blending
7. Simulating Natural Phenomena

At the end of the course the students have to present a project. Examples of subjects are:

- Mario Bros like.
  "Your program will enable the full 3D play and the editor ability for new levels of play."
- Star wars like.
  "With your program it will be possible to:
  pilot a space ship
  detect collisions with other ships or asteroids
  fight against other ships
  change the viewpoint, the textures, etc.

3.1.2. Image synthesis

This course pretends to introduce the whole of computer graphics. It is meant to cover the introductory course as presented in Hangzhou in 2004 [CHLS04]. Therefore its content is really broad:

1. Vision, Images and Colors
2. Image analysis, pixel based rendering
   This part uses the second and third chapter of Strothotte and Schlechtweg's book [TS02].
3. Fundamentals algorithms
   This part includes not only well known algorithms but also not so well known algorithms (for example Berstel's algorithm to compute straight lines).
4. Modeling
5. Rendering
   This part even includes NPR as a new way of render a scene.

At the end of the course a project has to be done by students. Examples of these projects are:

- Straight lines
  Test and compare the running times of the different programs seen in class (Bresenham's 1965, Bresenham's RLE, Castle and Pitteway's, Boyer and Bourdin's, Berstel's . . .).
- Dithering
  Use different dithering methods and compare the results.
- Painting
  Modify the pixel based rendering methods seen in class to obtain an impression of paint when applied on a photo.
- Toon shading
  Your program will render a scene through toon shading. It will then be applied on sequences of images and produce an animation with, if possible, visual continuity.

As one can imagine some of these projects seem now clearly outdated and are not frequently choosen by students.

In the figure 1 you can see a screen shot from such a work. Here it is possible to move the view point toward the statue or turn around it.

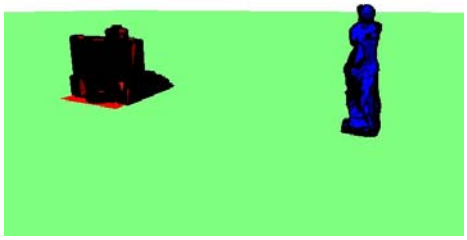

Figure 1: *A screen shot from a visualization project.*

3.1.3. GPU programming

This course intends to make student program on GPU. As this is constantly changing, the content is adapted every year. In 2007 it was:

1. Graphic Processing Units
2. GLSL
3. Toon shading (vertex shader)
4. Toon shading (vertex and pixel shader)
5. Texture mapping
6. Multi-texturing
7. Bump-mapping
8. Shadow volume

Examples of projects:

- Soap bubbles
  - Dynamically move the bubbles in a shader.
  - Destroy the bubbles after a period of time.
  - Add lateral wind.
  - Make a game with soap bubbles.
- Cars
  To a simple animation of moving cars, add
  - puddles of water, with reflections,
  - the rain and the waves it generates in the puddles of water,
  - the snow,
  - an impression of speed.
- Helicopter
  - Simulation of helicopter in flight.
  - Add image modifications to improve the speed feeling.
  - Add the effect of the rotor move and wind on water.
  - Add the effect of the rotor move and wind on grass.

The example presented in figure 1 has been upgraded by the student to pass this course. Table 1 presents the simple fragment program that enables to improve the frame per second rate of the program.

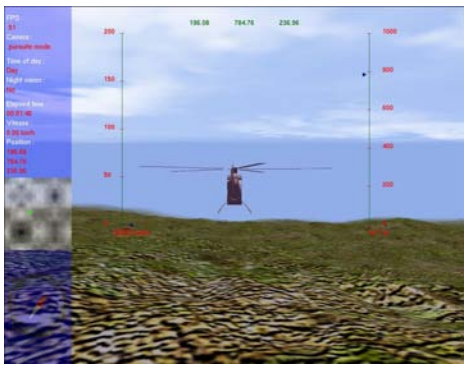

Figure 2: *A screen shot of a helicopter game.*

Images 2 and 3 present an helicopter game programmed by one of our students.

4. Pros and cons of the programming approach

We will now try to summarize some of the reasons teaching staff should or should not use programming as a good teaching approach.

4.1. Pros

- The main advantage of a programming oriented curriculum is that at any moment the students have the ability to test their own knowledge, to evaluate what exactly they know and are able to do. On the other side the teacher can evaluate the program to know the students have understood.

```
varying vec3 normal, lightDir;

void main()
{
  float intens;
  vec4 couleur;
  intens=(max(dot(lightDir,normalize(normal)),0.0));

  if (intens > 0.98){
    couleur = (gl_Color * 0.2);
  }
  else if (intens > 0.5){
    couleur = (gl_Color * 0.5);
  }
  else if (intens > 0.25){
    couleur = (gl_Color * 0.8);
  }
  else {
    couleur = gl_Color;
  }

  gl_FragColor=couleur;
}
```

Table 1: *A simple fragment code (couleur == color).*

- If one wants to to go farther than the step of *basic understanding*, programming is good way to assure a real *mastering* of the knowledge [BB06].
- Programming makes students work harder, specially in computer graphics. They spend hours to develop some part of the code, to obtain the "perfect" colors or interactivity.
- Programming increases the ability to concentrate. Most of the new generation, in France at least, has real difficulties to concentrate. They work a little and spend hours talking, sending and receiving messages about anything else. In their previous classes they never learned how to concentrate. Programming requires concentration and they learn it well.
- Our young generation lacks of the causality principle. Whatever the cause, for example it can be related to these politicians being declared "responsible but not guilty" even after casualties by the hundred, the fact is our students don't see the relation between action and consequences. With programming they have to understand that principle.
- Programming is a good way to acquire a sense of anticipation.
- Programming has paradoxically good results with students in situation of failure. We have, in our university, students who have failed earlier studies. Most of them think, at first, that they came here having nothing else to do. When we ask of them a lot of work as the only way to success most of them try this challenge. So they work a lot and the success is unavoidable. We'd say that with programming the success is more proportional to the work done than in any other kind of studies.

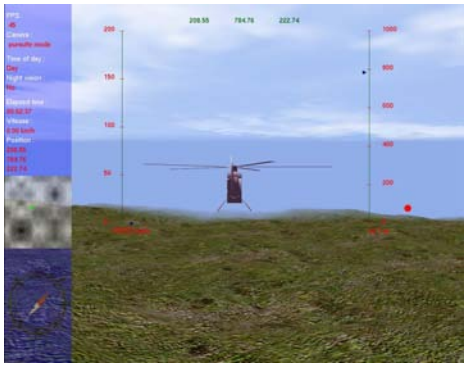

Figure 3: *Helicopter game higher view.*

4.2. Cons

- With programming, the learning goes slower. When one wants to present lots of material during a course, one can not make every student program everything. For example only one students programmed all the algorithms given to quickly compute straight lines (see above) in the three last years. What do the other ones remember of this class time?
- Programming requires harder work (as stated above). Therefore, in many cases, students are not able to do so much work and simply fail their studies. It is specially the case with students of low income families. They have to have a part time job to earn their living and a lot of home work is not always possible.
- Most times the program doesn't work perfectly and it's such a little error! It may take hours to fix a program even if one has already understood and make the most of it. Some students discourage at this point.

There is also the problem of *intuitive/rational* pedagogical methods. Programming is, in this case, both old fashioned and accurate: our domain deeply depends on rationality.

On the other hand, the greater part of pedagogy methods leads us to more intuitive methods of learning. The recent ways to teach even maths are less formal and more intuitive. One could also talk about PBL [MGJ06] as a way to follow a more intuitive path, less rational or anticipated. The main goal of the method is to improve results to get more students with a degree than with a failure in the end. It is not clear that this approach or even the intuitive approach achieve its goal of success. It seems to improve the statistics of any one course without improving the results of a whole curriculum.

5. Computer graphics programming

Now the discussion will be more closely focused on the computer graphics programming courses. The second course

(Image Synthesis) will not be discussed here. It is easily rebuilt as a usual lecture.

For the graphics programming course, the programming teaching technique has many advantages. Without it students would not learn how to use a modern API. It means also a better understanding of the maths used. It is very popular with students and gives the possibility to significantly improve other projects they have to do. For that purpose this class should be given at the end of the second year of our degree.

The main problems are that too much subjects have to be covered and therefore it is not possible to evaluate students on significant results on every subject.

For the GPU Programming course, the teacher in charge in the last years have noted some advantages/inconvenients.

- Some algorithms are too complex for third year students (multi-passes algorithms for example).
- The API course is too fresh and half of the students are not ready to take this new step. If 30 students passed the Graphics programming course, only 18 of them took the GPU class and only 12 of them were successful.
- The lecture on the GPU hardware has to be long and fastidious. The difference between geometry, vertex, rasterization, is not clear without being tried.
- This course gives an unique opportunity to use high level and low level programming.
- The results are interactive and very impressive therefore the course is popular on students.
- It is a good way to use more evolved maths (matrix inversion...).
- There is no need to recompile the program, the simple edition of the text file suffices to see real changes.

Two main questions remain. If the graphic programming course is done during the second year, will the GPU programming course be easier the following year? Should we use *glman* [BC07] or *CUDA* [CUD] to improve this course?

6. Conclusions

We have presented the curriculum of the bachelor degree, post Bologna requirements, taught in our university. This curriculum is programming oriented and the three computer graphics courses it includes specifically use this ability of our students. The advantages of a programming oriented curriculum have been discussed and more precisely the GPU programming course has been discussed. We don't offer an ending answer to the problems stated and hope a discussion at the conference will make us improve our teaching.

References

[BB06] BECKHAUS S., BLOM K. J.: Teaching, exploring, learning - developing tutorials for in-class teaching and self-learning. In *Education Programme at Eurographics 2006* (2006), Judy Brown W. H., (Ed.), vol. 25-4 of *Computer Graphics Forum*, pp. 840–841.

[BC07] BAILEY M., CUNNINGHAM S.: A Hands-on Environment for Teaching GPU Programming. In *SIGCSE Conference* (March 2007), SIGCSE, pp. 254–258.

[BCFH06] BOURDIN J.-J., CUNNINGHAM S., FAIRÉN M., HANSMANN W.: *Report of the CGE 06 Computer Graphics Education Workshop*. Tech. rep., EUROGRAPHICS/ACM-SIGGRAPH, September 2006. http://education.siggraph.org/conferences/eurographics/2006/cge-06-report-pdf.

[CHLS04] CUNNINGHAM S., HANSMANN W., LAXER C., SHI J.: The beginning computer graphics course in computer science. http://education.siggraph.org/conferences/eurographics/cge-04/Rep2004CGEworkshop.pdf, 2004.

[CUD] Cuda. NVIDIA. http://developer.nvidia.com/object/cuda.html.

[Cun99] CUNNINGHAM S.: Gve'99. http://education.siggraph.org/conferences/eurographics/gve-99/reports/papers/gve-fullreport.pdf, 1999. Report of the 1999 Eurographics/SIGGRAPH Workshop on Graphics and Visualization Education.

[Cun02] CUNNINGHAM S.: Cge 02 - final report, 2002. http://education.siggraph.org/conferences/eurographics/cge-02/report.

[FPA*06] FULLER U., PEARS A., AMILLO J., AVRAM C., MANNILA L.: A Computing Perspective on the Bologna Process. *ACM SIGCSE Bulletin 38*, 4 (December 2006), 142–158. http://www.cs.kent.ac.uk/pubs/2006/2447.

[MGJ06] MARTÍ E., GIL D., JULIÀ C.: A pbl experience in the teaching of computer graphics. *Computer Graphics Forum 25*, 1 (March 2006). Blackwell Publishing.

[TS02] T.STROTHOTTE, S.SCHLECHTWEG: *Non-Photorealistic Computer Graphics*. Morgan Kaufmann, May 2002.