# Do contests improve students skills in Computer Graphics? The case of API8

J.-P. Palus[1] and F. Belhadj[1] and J.-J. Bourdin[1] †

[1]LIASD, Université Paris 8, Saint-Denis, France

**Abstract**

*This paper presents a contest designed to improve the skills of students in Computer Graphics. The contest is adapted to the current skills of the students and uses a public graphic library. Students then have to produce a demo, generally a program which presents an animation. The main result presented in this paper is that with an appropriate set of tools, students program interesting demos to participate in the contest and their skills in Computer Graphics seem to improve significantly.*

**CCS Concepts**

• ***Social and professional topics*** → ***Computer science education; Student assessment;*** • ***Computing methodologies*** → *Animation; Rasterization;*

## 1. Introduction

Since the late 90s the teaching of Computer Graphics (CG) has drastically evolved. *OpenGL* changed a lot from version 1 to 2 and the deprecations enforced in the core profile of version 3.3 led to a greater complexity of use without a solid background. The assessment of a program is difficult. If for instance the parameters of the view point are wrong and the image does not show the scene, is the program a complete failure? As other authors have noticed, see for example the introduction of [BSP17], teaching *OpenGL* is increasingly difficult. The consequence is that libraries and tools are created to help students develop only the critical material. Teachers then have to avoid two main problems: presenting tools with the risk of students avoiding programming or teaching CG programming with the risk of a higher rate of failure and a subsequent decrease in enrollments. In our university we faced this dilemma and tried to solve it by launching a contest in Computer Graphics specially designed for our sophomore students. This was based on the idea of activity-led instruction [AP09]. While studies exist on the use of Robocup or other contests as teaching tools [AV02, SPS04] or even on assessment in programming [IAKS10,LDUC13], we did not find such work on CG. This paper will present the choices we made and Figures 3, 4 and 5 present some results of this experiment. The making of these images is presented in section 5.2.

## 2. Context

We needed to adapt the contest to our student audience. In our university students learn programming as freshmen (120h courses + 80h lab.) and carry on for their second year (same total). The other courses present machine architecture, basic courses in Logic, AI, Data Structures and Algorithmic. Most of these courses use programming as a way to learn. Students mostly program in C. We had no other choice than to respect this background as it makes cross-platform programming or implementing easy. We also wanted to use a suitable library because we did not want students to have to develop everything by themselves. We used **GLU/GLUT** [fre19], for years, but the current version is no longer functional with the core profile of *OpenGL* 3.3+.

Using **GLEW** [Gle19] or **GLFW** [Glf19] seemed possible but these libraries are now deprecated, not maintained or lacking easy-to-use functionalities. More complete libraries were available:

- **glGA** [PPGT14] is a geometric application framework. It runs on different platforms (Windows, Linux, MacOS) but is based on C++.
- Libraries like **S**haderLabFramework [TRK17] or **bRenderer** [BSP17] may be even better but are also based on C++.
- **GL4D** [GL419] is, to our knowledge the only multi-platform framework based on C providing easily accessible features under the *OpenGL* 3.3 core profile. It may be seen as an extension of **SDL2** [SDL19]: from rasterization based on *OpenGL* textures to higher level functions in 3D. For example, it is interesting to make students implement straight line algorithms such as Bre-

† Treasurer of EUROGRAPHICS



DOI: 10.2312/eged.20191028

senham's [Bre65] or Boyer's [BB99] but also a morphing or a polygon shading. We chose this framework in 2010 and were able to contribute to it. It is then the base of our contest and of our CG courses.

### 2.1. Technical context related to the selected library

Choosing GL4D allows for a progressive transition to the *OpenGL*'s core profile. Moreover, it provides simple tools to create demos as multiple *OpenGL* sub-programs drawing on the framebuffer in a transparent way.

Here, we summarize in a few points some of the features offered by this library to highlight how the transition was made, from teaching *OpenGL* 2 to 3.3 and more:

- The ability to create (on the CPU side), name (on the CPU side) and target (in shader code) matrices having the same functionalities as the matrices managed in the first versions of *OpenGL*:
  - `glMatrixMode(...)` becomes `gl4duBindMatrix(...);`
  - functions like `gl<Load*,Translate?,Rotate?...>` become `gl4du<Load*,Translate?,Rotate?...>`;
  - an automatic matrix stack management is accessible via `gl4duPushMatrix` and `gl4duPopMatrix`.
- The ability to create, through a single call, program shaders from files and/or strings. When shaders come from files, the library allows to dynamically update them.
- In the same way as GLUT, one can simply create and manage (`gl4duGen*/gl4dgDraw`) basic geometries such as: a quadrilateral, a sphere, a grid or a torus.
- The ability to apply screen/texture to screen/texture filters, such as: blur, sobel, median and operations.
- It is possible to associate "screens" and *OpenGL* textures to use basic 2D primitives such as `putpixel`, `getpixel` and `drawline`, with a total or a partial CPU/GPU synchronization.
- The ability to describe a timeline that synchronizes and plays *OpenGL* animation sequences (with asynchronous sound management and sound-image interaction) and makes transitions between sequences where the programmer mixes two textures at a time $t \in [0,1]$.

## 3. Computer Graphics Courses

In this section we present the first CG course of our curriculum and then summarize our other CG courses. Note that the course evaluation is still based on technical aspects, although the contest is a perspective for motivating our students, even these who do not participate.

### 3.1. The first CG course

The main goal of our first course is to present the field and to lead the students to program images and animations. The course starts with an introduction to the memory representation of an image and more specifically of a pixel. The implementation is focused on coding RGBA pixels using an unsigned 32-bit integer. Pixels are accessed via a 1D-indexed array. Shifts and binary masks are shown and used for reading and writing the color components. This introduction brings us to the sub-library of GL4D with 2D primitives and management of multiple `screens` based on the same image representation. The `screens` and the window can have different resolutions and linear or by-nearest-neighbor interpolations are presented.
After that, a simplified view of the GPU pipeline and the basics of data transfers between the CPU and the GPU are presented. The `screens` are textures on the *OpenGL* 3.3+ side and a default shader allows to map them on predefined 3D geometries (quad, sphere, cube, torus, ...). Initially, only quads are used to map `screens` to the viewport.
Starting with `putpixel` and `getpixel` routines, the lines and circles are revisited through re-implementations of Bresenham's 1965 [Bre65] and Bresenham's 1977 [Bre77] algorithms (one optional course goes further [BB99], especially concerning the DDA). Scan-line polygon filling is addressed and, in some years, the course extends to color gradient and texture mapping through a CPU implementation from-scratch.

As an application of the Bresenham's 77 algorithm, we usually propose to realize a discrete implementation of Voronoi diagram using growing circles (modifying primitives to manage conditional filling, addressing the *moiré* effect problem and using an image as a source of sites color). Figure 1 shows some results obtained after 4 or 5 course sessions, each session being 2 hours long.

In this context, our students discover for the first time what a program shader can be[†]. We focus on a simple fragment shader using 2D-unitary texture coordinates and data of `uniform` type. Thus, when applying a quad covering the viewport, a fragment is produced for each screen pixel. A computation of the Voronoi diagram by running through all the sites and choosing the color of the nearest one is given. Table 1 shows an implementation where diagram sites are sent as a 1D texture. Figure 2 presents the resulting Voronoi diagram for this GPU implementation. The top image matches the source code given in table 1, the other two use the distance to the second nearest site in order to shade the rendering.

The course continues with an introduction to animation based on examples using simplified models of Newtonian physics including collisions. We emphasize the importance of the elapsed time in the movement computations between two frames. Audio management is presented up to an introduction to spectral analysis and image/audio interaction. After that, we introduce the GL4D `DemoHelper` module, briefly detailed in the next paragraph, that allows managing multiple animations and transitions between them. The course usually concludes with a few introductory sessions to 3D: the use of GL4D solids and 3D matrices (view, model and projection).

[†] It would be interesting to present an implementation that sends triangulated cones centered on the sites coordinates and puts into practice the z-test. However, at this stage, students knowledge in 3D modeling (including vertex arrays), transformations (scale, translations and projection) and *OpenGL* (including options and buffers) is insufficient to deal with this variant with serenity.

```
#version 330
uniform sampler1D sites;
uniform float step;
in  vec2 vsoTexCoord;
out vec4 fragColor;
vec4 voronoi(vec2 xy) {
  float i, minidx = 0.0, d;
  float mindist;
  mindist=length(xy-texture(sites,minidx+step).xy);
  for(i = 2.0 * step; i < 1.0; i += 2.0 * step) {
    vec2 position = texture(sites, i + step).xy;
    if(mindist > (d = length(xy - position.xy))) {
      mindist = d;
      minidx = i;
    }
  }
  return texture(sites, min_idx);
}
void main(void) {
  fragColor = voronoi(vsoTexCoord);
}
```

**Table 1:** *A GPU Voronoi diagram as an introduction to fragment shaders.*

**Figure 1:** *CPU implementation of the Voronoi diagram. The image at the bottom was realized by a sophomore student. Colors are selected on a photography as sites colors to initialize the diagram (zoom in on the image to see the detail). At each frame, depending on the sites placement, the effect can be more or less interesting.*

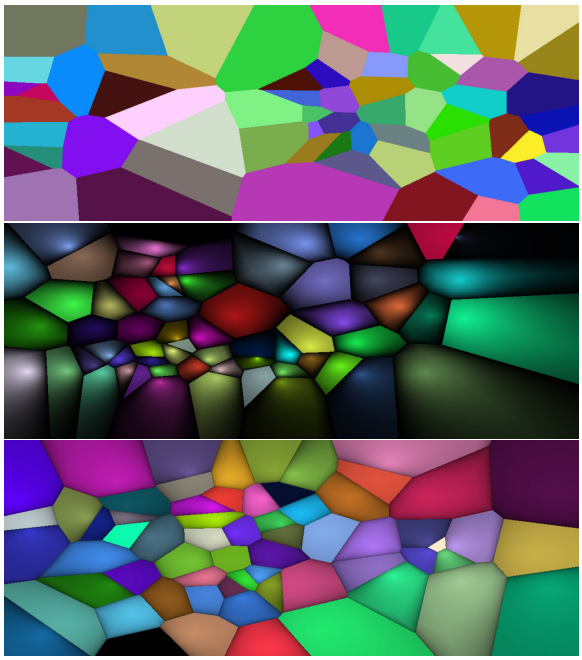

**Figure 2:** *GPU implementation of the Voronoi diagram.*

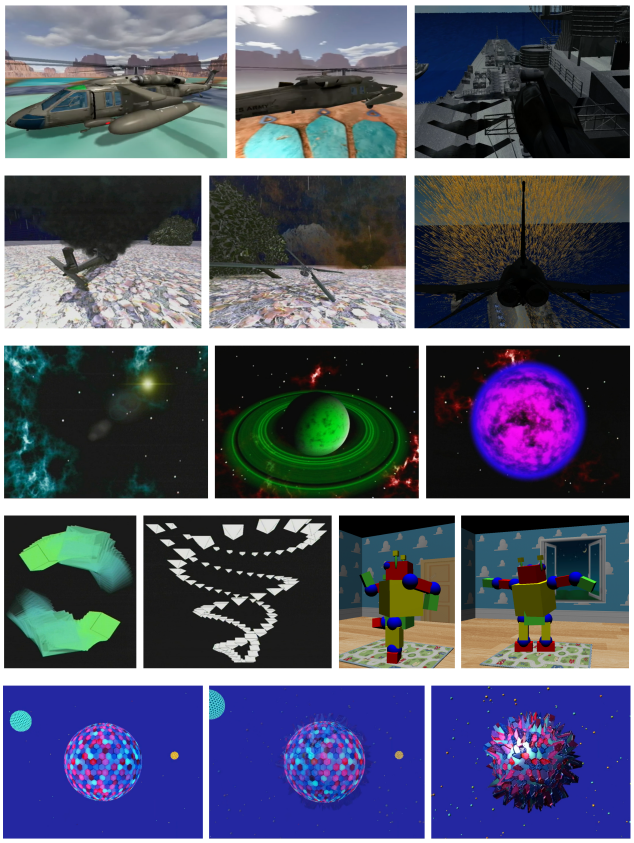

**Figure 3:** *Examples of images extracted from demos.*

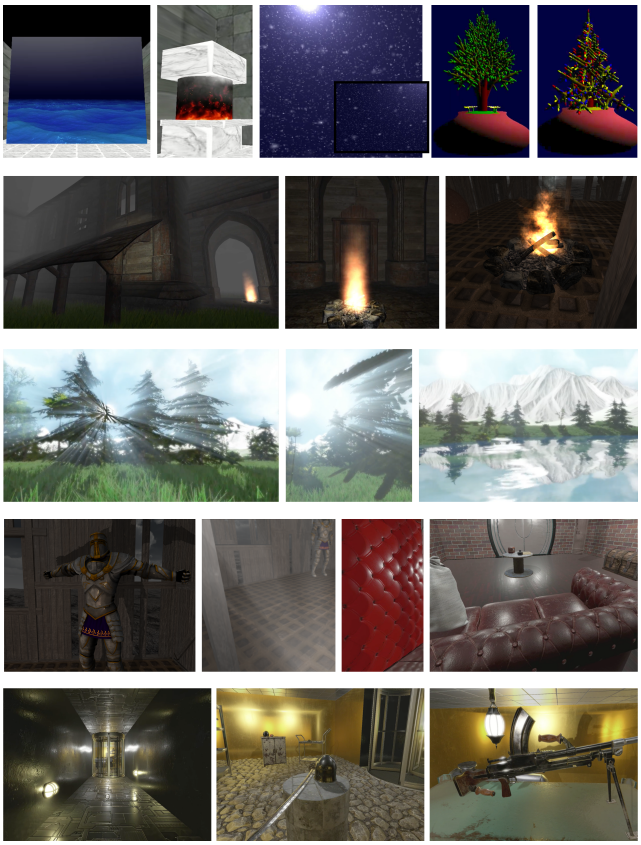

**Figure 4:** *Examples of achievements made by more advanced students.*

### 3.2. The DemoHelper module

To be able to program a proper animation, students need to get higher level functions. We propose the use of the GL4D `DemoHelper` module, a set of functions to build independent sequences with their own data sets including sound and a transition functions between sequences without a break.

The structure proposed in Table 2 is used to describe an animation or a transition. Each one starts at a given time and has 4 states allowing initialization, updating data according to the sound, drawing or releasing data before leaving. A transition plays two animations by gradually mixing them with the elapsed time to go from one to the other.

Table 3 shows how to realize an animation based on 2D primitives which can adapt to the sound. The use of static variables gives the ability to leave the animation data accessible from one frame to the other but also from one state to another. It should be noted that the state `GL4DH_UPDATE_WITH_AUDIO` is produced by a callback. This callback is generated by the process managing the audio that is different from the main process (the one creating the *OpenGL* context). Thus, the audio process can compute a radius that will be applied to the disc at the drawing time (*ie*, during another call). On this basis, we explain to students how to modify their graphics programs in order to integrate them according to this model.

**Figure 5:** *This work, done by one of our students, illustrates the use of linear filters implemented in GPU to have a real-time processing over the framebuffer's output in order to obtain artistic renderings.*

```
enum GL4DHstate {
  GL4DH_INIT,
  GL4DH_UPDATE_WITH_AUDIO,
  GL4DH_DRAW,
  GL4DH_FREE
};
struct GL4DHanime {
  Uint32 time;
  void (* first)       (int state);
  void (* last)        (int state);
  void (* transition)  (void (* anim0)(int),
                        void (* anim1)(int),
                        Uint32 fullTime,
                        Uint32 elapsedTime,
                        int state);
};
```

**Table 2:** *The DemoHelper main structure.*

### 3.3. The other CG courses

Furthermore, the other CG courses also use basic or advanced functionalities of GL4D:

- Image Synthesis, for junior students: it is a theoretical and practical approach to DDA, to image processing and to expressive rendering of an image (hatching, painting...).
- Basic GPU programming, for junior students: it is the first full course on the new *OpenGL/GLSL* architecture (GL primitives, vertex arrays, texturing, lighting, geometric distortions... are detailed).
- Advanced GPU programming and computer vision (using OpenCV [OPE19]), for master students: here, data generation,

```
void audio_disk_anim(int state) {
  Sint16 * stream;
  GLint acc, length, i;
  static GLuint screen_id, radius;
  switch(state) {
  case GL4DH_INIT:
    screen_id = gl4dpInitScreen();
    gl4dpUpdateScreen(NULL);
    return;
  case GL4DH_FREE:
    gl4dpSetScreen(screen_id);
    gl4dpDeleteScreen();
    return;
  case GL4DH_UPDATE_WITH_AUDIO:
    stream = (Sint16 *)ahGetAudioStream();
    length = ahGetAudioStreamLength() » 1;
    for(i = 0, acc = 0; i < length; i++)
      acc += abs(stream[i]) » 4;
    radius = 100 + acc / length;
    return;
  default:
    gl4dpSetScreen(screen_id);
    gl4dpClearScreen();
    gl4dpSetColor(RGB(255, 255, 255));
    gl4dpFilledCircle(gl4dpGetWidth() » 1,
                      gl4dpGetHeight() » 1,
                      radius);
    gl4dpUpdateScreen(NULL);
    return;
  }
}
```

**Table 3:** *Example of an animation that modulates a disc radius according to the audio stream.*

advanced modelling, rendering techniques and image processing are addressed.

## 4. The contest

The API8 [API19] contest was launched in June 2014 with late and little communication support. Since then, the contest has always taken place between one and two months after the end of the 2nd semester.

Now students start to work on the contest before the end of the semester and therefore their work for the contest is presented as their main result for the Computer Graphics course. The assessment process is fulfilled by a program committee. In 2014, the program committee was composed by an equal mix of CG teachers, computer science teachers and academic staff. The following year, it was fully composed of children. Since 2016, half of the program committee is composed by teachers and the other half is composed by externals particpants and non academic staff and children. Therefore, aspects such as technical, storyline and visual aesthetic are balanced. The contest is funded by our university and local authorities and valuable prizes are given to the best three works. As a lively performance each work is to be compiled and executed on a Linux system proposed by the organizers.

Moreover, since the second edition, other computer science fields have joined the contest including an "AI Chatterbot" track (since 2015) and an "AI for StarCraft2" track (since 2017).

| | 2014 | 2015 | 2016 | 2017 | 2018 |
|---|---|---|---|---|---|
| Sophomores | 50 | 71 | 57 | 69 | 60 |
| No work given | 17 | 34 | 20 | 29 | 27 |
| Unsatisfactory marks | 16 | 13 | 6 | 3 | 9 |
| Satisfactory marks | 17 | 24 | 31 | 37 | 24 |
| Contestants | 12 | 10 | 6 | 15 | 14 |

**Table 4:** *Quantified results per recent years*

## 5. Results

### 5.1. Quantified results

Table 4 presents some quantified results for the CG course of our curriculum. In 2014 this course took place during the third year and since 2015 during the second year of our bachelors degree. We do not present results from earlier years. The first two years, 2014 and 2015, students did not associate the contest with the course and thought they were expected to work on two different projects. We choose to simplify the table with only two categories (Unsatisfactory or Satisfactory work), the number of very good marks should have a meaning too. The last row is the number of contestants but a few of them come from other universities.

Our average cohort is around 60 sophomores. One must note that if a student missed one or more courses during the second year this student is counted as both a sophomore and a junior. Therefore some students that gave no work have passed on previous years. Some other students have failed the first semester: they are nonetheless registred for second semester courses they will not attend. The last limit of our numbers is that this course is not mandatory. From the management point of view it is easier to enroll students in each course they may or may not choose. Therefore the "No work given" values do not directly reflect a failure from students in the course.

To decipher the discrepancy between the number of students registered in this course and the number of works received we asked other colleagues about their numbers. For one mandatory second semester course, the number of registred students has been 73. 34 students presented their work and 8 students attended the class but did not present their work. Other results are similar. We still have no way of knowing how many of the registred students intended to work for the course.

A surprising result of the contest is that a lot of students prepare the contest, present they work as their assessment but don't present it in the contest. As one can see, the number of Satisfactory marks greatly increases as students work for the contest. Having more than 30 successes is an improvement over the previous years. In itself such a result is a sufficient argument to implement a contest in CG if one's pass rates are too low. But the main result is the drop of unsatisfactory works.

**Remark:** Year 2018 is an exception since the university was on strike and therefore closed to both students and staff for seven weeks during the second semester. It is very difficult to come back

to work after almost a two months long break. We were surprised to see more than half of the sophomores coming back to follow the course. Because of this situation we think the results for 2018 are not comparable to the results of previous years we included them anyway.

Table 5 presents the pedagogic summarized data.

| | |
|---|---|
| Summary | API8 is a contest to motivate students for their first CG course. They learn to program a demo. |
| Learning Outcomes | Program a demo. Combine sound with animation. |
| Classification(s) | Social and professional topics -> Computer science education; Student assessment; Computing methodologies -> Animation; Rasterization; |
| Audience | Second year students, sophomores. |
| Dependencies | Skills in C programming are mandatory, algorithmics and data structures may come in handy. |
| Prerequisites | First main assignment in Computer Graphics. |
| Strengths | To be able to present a whole project including images, animations and sound. The competition. |
| Weaknesses | Its time consuming counterpart. |
| Variants | The main scenario of the animation is their own. The teacher may be more or less helpful depending on the level of students enrolled. |
| Assessment | Visual aesthetics; quality of the animation; story told if any. |

**Table 5:** *CGEMS Metadata for API8.*

### 5.2. More results

We present in Figure 3 some images as examples of achievements made by our undergraduate students and presented during previous sessions of the contest. These demos backed up and extended concepts seen during computer graphics courses. The first two lines show works exploiting models loading with or without textures, Gouraud or Phong lighting and the use of particle systems on the CPU side. The third line shows an embodiment highlighting the use of textures and transparency to obtain light and atmosphere effects. The following line shows extensive uses of the transformation matrices and the matrix stack proposed in GL4D. The last one illustrates transitions writing (the demoHelper module of GL4D exploits the framebuffer to play two animations simultaneously and to help mix them) and the use of the geometry shader in mesh deformations according to the sound (also with the help of the demoHelper module).

Figure 4 presents examples of achievements made by our undergraduate students and one masters student (a former undergraduate). Here, students have proposed and independently explored techniques not seen in class. The top line shows an example of work presented at the *64k track* where we can see a procedural generation of sea, another of flames (both in GPU) and an engine for L-Systems generation. The following lines show very complete achievements including volumetric fog and lightings, GPU particle systems for flame generation, environment mapping, steep parallax mapping and more complex lighting models including the bloom lighting.

### 5.3. Stunning results

We are mostly interested in numbers, but subjective evaluations may be meaningful. The first remark is that the contest takes place, each year, after the end of the semester in order not to penalize the other courses.

We have been surprised to notice that students are still working on CG after the end of the semester. The main result for us, as teachers, is the number of domains that were never approached by our students, even the very best of them, that are now treated and added to their work:

- Animations.
- Realistic movement of joints.
- Camera movements.
- Artistic view of their work. How many times did you receive images no better than what children give to their parents? Now we receive works with a real artistic content.
- Coordination of the sound and movements in the image. It induces students to work more on the image.

Therefore, the results to the Computer Graphics course are better now.

### 5.4. Students' point of view

Some of our students are largely below average students. One failed to the Algorithmic course twice. After being involved in the contest, this student started to work better and succeeded in Algorithmic during the following semester. Another one had real difficulties with programming. His first participation in the contest was marked by an average but well scripted demonstration that produced several "segmentation fault" which became his nickname. This experience changed this student into a valuable one who is currently working as programmer in a well known Computer Graphics company. Other students were on the top side and their results were not as surprising. But it is always a pleasure when sophomores come with a 3D shader based demo one year before the extended course on GPU programming, or implement an augmented reality system with a virtual face that synthesizes an AIML-bot speaking in a real scene.

Most of our students present their demo when they apply for internships or for a job. They all say it was a major asset to be accepted. Some of them got internships and then jobs in CG or video games companies.

## 6. Conclusion

For us the main contribution of this experiment is to encourage staff to develop their own contest or to encourage their students to participate ours. In any case it seems that the contest increases student motivation in our Computer Graphics courses.

**Acknowledgments**

The contest was funded by *Université Paris 8* – UFR MITSIC and we would specially like to thank *Plaine Commune*, *audio-3D*, *Grafeet* and *MFA* for funding this contest and the students that participate to our courses and/or the contest.

**References**

[AP09] ANDERSON E. F., PETERS C. E.: On the Provision of a Comprehensive Computer Graphics Education in the Context of Computer Games: An Activity-Led Instruction Approach. In *Eurographics 2009 - Education Papers* (2009), Domik G., Scateni R., (Eds.), The Eurographics Association. 1

[API19] Api8, 03 2019. Accessed 03.03.2019. URL: http://www.api8.fr. 5

[AV02] AHLGREN D., VERNER I.: An international view of robotics as an educational medium. In *International Conference on Engineering Education* (August 2002). Manchester. 1

[BB99] BOYER V., BOURDIN J.-J.: Fast lines: A span by span method. *Comput. Graph. Forum 18*, 3 (1999), 377–384. 2

[Bre65] BRESENHAM J.: Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965), 25–30. 2

[Bre77] BRESENHAM J.: A linear algorithm for incremental digital display of circular arcs. *Commun. ACM 20*, 2 (1977), 100–106. 2

[BSP17] BÜRGISSER B., STEINER D., PAJAROLA R.: bRenderer: A Flexible Basis for a Modern Computer Graphics Curriculum. In *EG 2017 - Education Papers* (2017), Bourdin J.-J., Shesh A., (Eds.), The Eurographics Association. 1

[fre19] Freeglut, 03 2019. Accessed 03.03.2019. URL: http://freeglut.sourceforge.net/. 1

[GL419] Gl4d, 03 2019. Accessed 03.03.2019. URL: https://github.com/noalien/GL4Dummies. 1

[Gle19] Glew, 03 2019. Accessed 03.03.2019. URL: http://glew.sourceforge.net/. 1

[Glf19] Glfw, 03 2019. Accessed 03.03.2019. URL: https://www.glfw.org. 1

[IAKS10] IHANTOLA P., AHONIEMI T., KARAVIRTA V., SEPPÄLÄ O.: Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2010), Koli Calling '10, ACM, pp. 86–93. 1

[LDUC13] LI C., DONG Z., UNTCH R., CHASTEEN M.: Engaging computer science students through gamification in an online social network based collaborative learning environment. In *International Journal of Information and Education Technology* (01 2013), vol. 3, pp. 72–77. 1

[OPE19] Opencv, 03 2019. Accessed 03.03.2019. URL: https://www.opencv.org. 4

[PPGT14] PAPAGIANNAKIS G., PAPANIKOLAOU P., GREASSIDOU E., TRAHANIAS P.: glGA: an OpenGL Geometric Application Framework for a Modern, Shader-based Computer Graphics Curriculum. In *Eurographics 2014 - Education Papers* (2014), Bourdin J.-J., Jorge J., Anderson E., (Eds.), The Eurographics Association. 1

[SDL19] Sdl2, 03 2019. Accessed 03.03.2019. URL: https://www.libsdl.org. 1

[SPS04] SKLAR E., PARSONS S., STONE P.: Using robocup in university-level computer science education. *J. Educ. Resour. Comput. 4*, 2 (June 2004). 1

[TRK17] TOISOUL A., RUECKERT D., KAINZ B.: Accessible GLSL Shader Programming. In *EG 2017 - Education Papers* (2017), Bourdin J.-J., Shesh A., (Eds.), The Eurographics Association. 1