Redesign of an Introductory Computer Graphics Course

Philipp Ackermann and Thomas Bach

InIT, School of Engineering, Zurich University of Applied Sciences (ZHAW), Winterthur, Switzerland

Abstract

The redesign of our historically grown Computer Graphics course was primarily triggered by the need to incorporate modern, shader-based OpenGL. This technical modification led to discussions on the relevance of course topics, the order of presentation, the role of sample programs, and problem sets addressed in lab exercises. The redesign resulted in changing from a bottom-up to a top-down approach and in a shift from low-level procedural OpenGL to the use of a high-level object-oriented 3D library on top of WebGL. This paper presents our motivation, applied principles, first results in teaching the redesigned course, and student feedback.

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

1. Background and Context

At ZHAW School of Engineering, Computer Graphics (CG) is an elective module with 4 ECTS credits in the Bachelor program of the Computer Science curriculum. Being a university of applied sciences (technical college) our students are aged between 20 and 25. As an introductory course the educational goals cover the knowledge transfer of basic CG theory as well as gaining practical experience in developing CG software. The course was taught for over 10 years and followed in form and content the various editions of the widely accepted introductory book "Interactive Computer Graphics" by Angel and Shreiner [Ang00, AS12]. While focussing on 3D graphics the course mainly used the OpenGL technology for sample codes and lab exercises in C/C++.

Week	Lecture	Lab Exercises
1	CG & OpenGL Introduction	Dev. Env. Setup
2	Mathematical Foundation	OpenGL
3	Polygonial Meshes	dito
4-5	OpenGL Programming	dito
6-7	Shading and Textures	dito
8	Raytracing	Java Raytracer
9-10	Animation & Collision Det.	dito
11-12	GPU Prog., OpenGL Ext.	GLSL
13-14	Color, Curves & Surfaces	dito

Table 1: *Overview of original CG course.*

2. Changing Requirements

Although theoretical topics in computer graphics are commonly accepted and stable, technology innovations have a strong impact on used hardware and software within the computer graphics industry. In the past this evolution led to a modern, shader-based graphics pipeline and the fixed pipeline of early OpenGL became deprecated. In our Computer Graphics course some lectures and lab exercises for shader programming were successively added. However this led to a confusing mix of deprecated fixed pipeline and modern shader-based OpenGL source code. We therefore needed to revise the historically grown sample code base within our lectures and exercises.

In the current Internet age knowledge can be obtained in much better ways than sitting passively in a classroom. Today's students have instant access to online books, good presentations and tons of code samples on computer graphics topics. Nowadays teaching at a school has to be more than transmission of (testable) information. Besides guidance through relevant topics by classroom lectures, we wanted to provide an active and interactive learning environment. The goal of our redesign was therefore a well-balanced mix of theory and learning by doing through interactive samples and attractive lab assignments.

Besides these technological adjustments we became aware of the importance of meaningful content [CC09] to be graphically presented in sample applications and in lab exer-

DOI: 10.2312/eged.20151021

cises. Presenting realistic aspects of applied computer graphics in an interesting context is key for students' motivation and learning success. Students that grew up with sophisticated computer games and 3D visual effects in movies have high expectations on interactivity and attractive content. We therefore wanted to shift from a pure technology focus to a broader approach that includes programming skills as well as experience in handling rich 2D and 3D media (e.g., gaining know-how by applying corresponding computer-aided design tools and 2D/3D file formats).

3. Redesigned Computer Graphics Course

The following chapters give reflections on applied principles for redesigning our introductory Computer Graphics course. First we present our teaching experience. Afterwards we evaluate the students' feedback. Our students are software engineers and have typically a professional background with a strong application focus. In previous semesters they were taught in linear algebra and software engineering (among others), so they have considerable knowledge in both applied mathematics and software development.

The redesigned course (see overview in Table 2) was conducted in fall of 2014 from mid September to end of December. It consisted of 14 two-hour lectures plus weekly two-hour lab sessions as part of the 5th semester of the 3 year Bachelor studies program. The course assessment included weekly reviews of student lab exercises and a written examination at the end of the semester. The final grade was calculated by the result of the exam (60% weight) and the amount of lab exercises finished successfully (40% weight).

Week	Lecture	Lab Exercises
1	CG Introduction & History	Web Dev. Env.
2	2D Computer Graphics	SVG/SVG.js
3	Information Visualization	D3.js
4-9	3D Computer Graphics	Three.js
10-11	GPU Programming	GLSL/WebGL
12-14	Raytracing, Color Theory	Blender Plug-in (with Python)

Table 2: *Overview of redesigned CG course.*

The redesigned course was attended by 33 students and held in two classes, one with full-time students (62%), the other with part-time students (38%). The evaluation on students' preferences were conducted via anonymous online voting during lectures. The authors taught the redesigned course as well as the old one a year ago, therefore the presented results are not biased by different instructors.

3.1. High-level 2.0

Early OpenGL was seen as a high-level library because it abstracted low-level rasterization methods and supported immediate mode rendering in a fixed graphics pipeline. Modern OpenGL is more flexible but as a side effect lowers the supported abstraction level. Early OpenGL sample applications already needed quite a lot of utility libraries to provide window management, event handling, and file I/O for images and geometry. It took a great deal of effort to support and teach these utility libraries, especially in a multi-platform environment. Modern OpenGL needs even more additional libraries, e.g., for matrix transformations and shader programs, and it became significantly more complex to develop a minimal 3D program on top of modern OpenGL.

The redesigned CG course is taught at the Zurich University of Applied Sciences where practical relevance of students' know-how and hands-on experience are important. Based on our experience with industry partners it seems to be more likely that our students will work with an already established graphics framework rather than building their own graphics library on top of OpenGL from scratch. On the other hand high-level graphics libraries such as in-house developed frameworks, open-source visualization toolkits (e.g., VTK) or commercial game engines encompassing object-oriented structures for 3D views, scene graphs, animations, etc. are typical in professional IT environments.

Because our aim is to teach industry-relevant CG concepts with attractive assets we emphasize higher abstractions on the technology stack (high-level 2.0) by additionally including high-level libraries, applications to create graphical content, as well as file formats to exchange media assets.

Due to these reasons it was clear that the redesigned CG course will need additional software functionality on top of modern OpenGL. The following solutions were evaluated:

- Open-source educational C/C++ framework such as glGA [PPGT14] of the University of Crete or the framework from the University of Stuttgart [RME14]
- Closed-source commercial game engine such as Unity
- Web-based JavaScript environment on top of WebGL

Although the educational glGA framework possesses attractive components and would have been a reasonable continuation of our former course, a Web-based JavaScript environment was chosen due to the following reasons:

- WebGL gains momentum and is supported on (mostly all) desktop and mobile Web browsers
- The Browser-based environment solves the multi-platform requirement (Win, Mac, Linux) as well as window and event handling
- Web-based JavaScript libraries are open-source, free and often supported by an active developer community
- The newest edition of the introductory text book [AS14] moved from OpenGL to WebGL
- It is easy to integrate text, images, 2D graphics, 3D graphics as well as UI elements into rich Web applications
- Our students were already introduced in Web technology so our course can concentrate on CG topics

By moving to a Web-based JavaScript programming environment, we were able to shift the focus from low-level procedural OpenGL to an object-oriented 3D framework. We have chosen the Three.js [Dir13, Par14] JavaScript library built on top of WebGL. Three.js provides modular extensions for navigation, file import/export, animations, collision detection, and more. In addition the COLLADA file format (an open standard for 3D content exchange) and Blender as an open-source multi-platform 3D content editor (with JSON as exchange format) are covered in lectures and exercises in order to handle attractive 3D assets.

In parallel we set up a similar technology stack for 2D computer graphics (Table 3). The SVG standard is promoted for file exchange and as Document Object Model (DOM) for 2D graphics in the Web browser. The JavaScript libraries SVG.js and D3.js [BOH11, Mur13] are utilized to create interactive 2D graphics and information visualizations. The open-source application Inkscape provides a multi-platform editor for creating 2D graphics content.

	2D	**3D**
File Format	SVG	COLLADA
Application	Inkscape	Blender
Library	D3.js	Three.js
API	SVG.js	WebGL/OpenGL
Driver	Display Driver	Graphics Card Driver
Hardware	Frame Buffer	GPU, Depth Buffer

Table 3: *Covered topics of 2D and 3D technology stack.*

By using high-level libraries and content-rich media assets, it is assumed that students understand the basic principles of computer graphics in an easier and much engaging way. Learned concepts understood within the Three.js JavaScript library prepare students to apply computer graphics concepts in Web applications as well as in larger 3D frameworks and game engines.

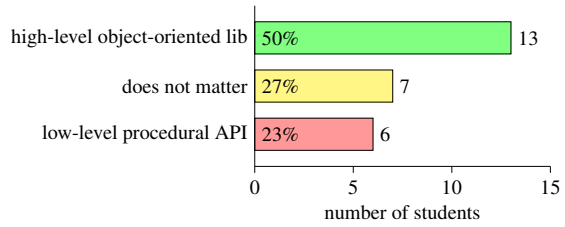

Figure 1: *Students' preferences on abstraction level.*

Feedback from students is positive on the high-level approach of the redesigned course (Figure 1). Critical votings with preferences for low-level procedural API came from some full-time students and were mainly due to the course announcement highlighting OpenGL as in earlier years and some bias against the JavaScript programming language.

3.2. Radical Top-down Approach

As in most CG classes at universities, our former course followed a bottom-up approach where mathematical topics such as vector algebra and transformation matrices were intensively examined at the beginning of lectures and exercises. Together with the low-level API approach of OpenGL, it took several weeks within the semester before any interesting computer-generated graphics appeared on the screen (see Table 1). Using a high-level technology approach we were able to alter the order of presenting CG topics. By applying a radical top-down approach, the subsection on 3D computer graphics starts content-driven with geometry in 3D file formats, scene graph concepts and sample applications using Three.js. By working with real data within a 3D editor and by operating sample Web3D applications, students apply 3D technology right from the beginning and get a first intuition on CG topics. Step-by-step more details are explained during the CG course, covering light and material models, transformation and projection matrices, texture mapping, GPU hardware, shader programming, etc. (Table 4).

W	**Lecture Topics**
4	Scene Graph, Camera, Geometry, Three.js Intro
5	Light, Color, Modeling in Blender, 3D File Formats
6	Math Basics, Transformation & Projection Matrices
7	Interaction & Navigation in 3D
8	Environment, Bump & Shadow Mapping
9	Animation, Collision Detection
10	GPU Hardware, OpenGL, WebGL
11	Shader Programming with GLSL

Table 4: *Details of 3D Computer Graphics subsection.*

Such a top-down approach is more realistic in how challenges are tackled in practice, seems to be more engaging and therefore positively supports self-motivation among the students.

3.3. Explorative Sample Source Code

Sample applications are key to facilitate knowledge acquisition a) via "learning by simulating" concepts in interactive examples and b) via "understanding by reading" corresponding source code. Web-based sample programs are not primarily used for ease of deployment but to provide an easy to use programming playground that runs on multiple platforms (Win, Mac, Linux). Beside the lectures' presentation slides, about 45 2D and 40 3D sample applications are provided to our CG students. The demo programs are single file programs (no include files except used libraries) that encompass HTML5 declarations and JavaScript code. The demo files have a size of 50 to 400 lines of code (including comments and blank lines) and present straightforward basic CG concepts. The demo source code is therefore easy to

present during lectures and tightly arranged to read without switchovers in a text editor.

Technol.	Topics of Demo Programs
Canvas	line, bezier curve, sprite anim, rotary gauge
SVG	basic shapes, gradient colors, transparency, bezier path, text, image, blur & shadow filter
SVG.js	event handling, interaction & animation
D3.js	charts, network graph, map, data filtering
Three.js	projection, wireframe, material, light, GUI, model transformation, camera navigation, geometries, CSG, animation, shadow, fog, texture mapping, picking, dragging, handles, file I/O via COLLADA,, JSON, OBJ, VTK
GLSL	phong shading, normal & parallax mapping

Table 5: *Overview of demos provided as source code.*

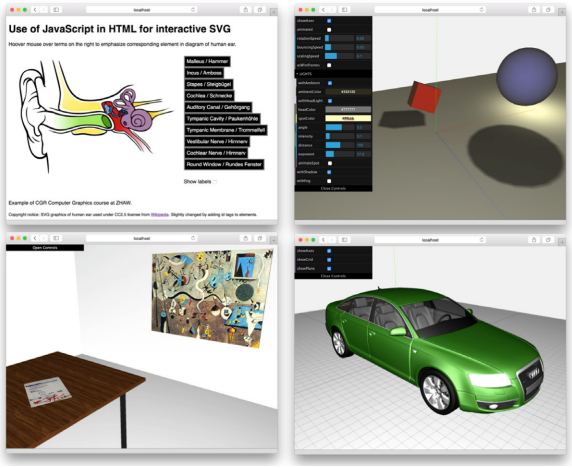

Figure 2: *Examples of Web-based demo programs.*

The demo programs with their open source code were well received by the students (Figure 3) and served as starting point for their lab work. The Web-based demo programs are available for download at `http://github.engineering.zhaw.ch/VisualComputingLab/CGdemos`.

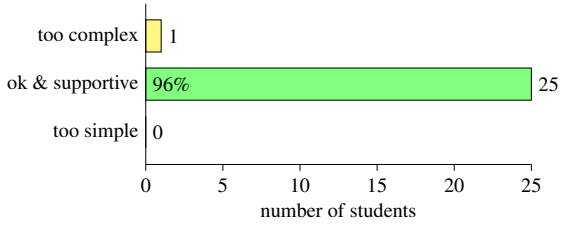

Figure 3: *Students' evaluation on sample code.*

3.4. Open Scope of Lab Exercises

Some of the lab exercises are given as open scope projects. For example students were asked to create interactive information visualization of data of their own interest, e.g., from their hobby or professional background or by using Open Data sources (Figure 4). As a 3D project, the task was to create a 3D product configurator (Figure 6) of a self-selected artifact. By assigning such content-driven problem sets, students learn how to transfer basic CG concepts to convincing solutions. They gain experience in managing 2D and 3D assets using content editors, (free/open) data or model bases, and corresponding exchange file formats. Furthermore, students learn how to integrate 2D/3D content with interactive functionality to create attractive Web applications. Most of the students appreciated the open scope of lab work (Figure 5) and were therefore inspired and motivated.

Figure 4: *Examples of students' lab work on 2D information visualization using the D3.js JavaScript library.*

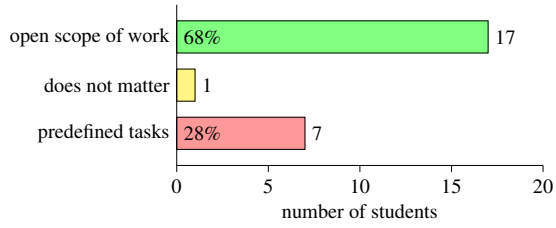

Figure 5: *Students' preferences on lab work.*

4. Results and Conclusion

Figure 7 shows that students in the redesigned course achieved better results in their excercises. First of all they were able to execute additional tasks on 2D SVG graphics, interactive information visualization with D3.js, and 3D

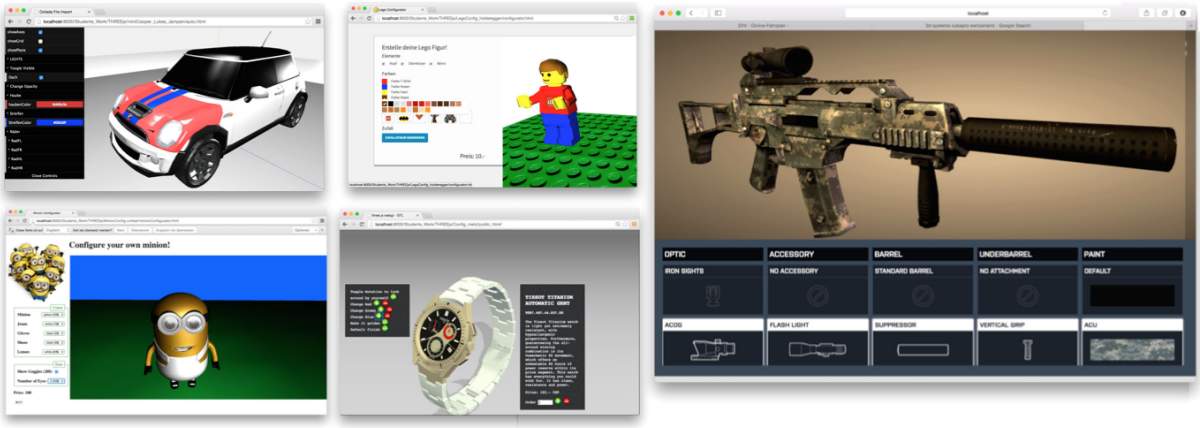

Figure 6: *Examples of students' lab work on 3D product configuration using the Three.js JavaScript library.*

modeling with Blender. Secondly students' performance in their lab work on 3D graphics (OpenGL/Three.js 1 & 2) improved substantially shown by an increased return rate of finished exercises. Our students often skip the last exercises, either because they already have enough points or because they are too absorbed by other courses. We therefore switched the order of the last two lab exercises, so that more students practice GLSL shader programming at the expense of gaining experience in developing a raytracer.

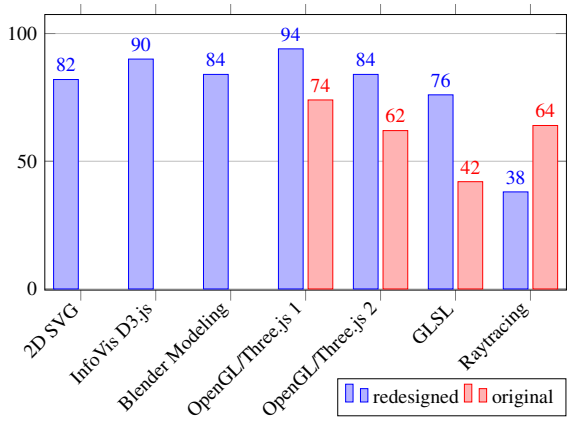

Figure 7: *Rate of finished lab exercises in %*

Due to the high-level, top-down approach of the redesigned Computer Graphics course we gained

- a broader scope, adding 2D graphics and Information Visualization to the instructed CG topics
- more interactive sample applications with source code
- rich 2D/3D content right from the beginning of the course
- experience in creating 3D models and animations
- motivated students that realized attractive projects valuable for their own portfolio.

Feedback via lab journals acknowledges the improvements. Due to these positive results we plan to apply this concept to further Visual Computing courses within our curriculum.

References

[Ang00] ANGEL E.: *Interactive Computer Graphics: A Top-down Approach with OpenGL.* Addison-Wesley, 2000. 1

[AS12] ANGEL E., SHREINER D.: *Interactive Computer Graphics: A Top-down Approach with Shader-based OpenGL.* Pearson international edition. Addison-Wesley, 2012. 1

[AS14] ANGEL E., SHREINER D.: *Interactive Computer Graphics: A Top-Down Approach with WebGL.* Pearson Education, Limited, 2014. 2

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (Dec. 2011), 2301–2309. doi:10.1109/TVCG.2011.185. 3

[CC09] CASE C., CUNNINGHAM S.: Teaching computer graphics in context. *Computer Graphics Education 09 Workshop* (2009). URL: http://media.siggraph.org/education/reports/CGE09-Workshop-Report.pdf. 1

[Dir13] DIRKSEN J.: *Learning Three.js: The JavaScript 3D Library for WebGL.* Packt Publishing, 2013. 3

[Mur13] MURRAY S.: *Interactive Data Visualization for the Web.* O'Reilly, Sebastopol (CA), 2013. 3

[Par14] PARISI T.: *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages.* O'Reilly Media, 2014. 3

[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. doi:10.2312/eged.20141026. 2

[RME14] REINA G., MÜLLER T., ERTL T.: Incorporating modern opengl into computer graphics education. *Computer Graphics and Applications, IEEE 34*, 4 (2014), 16–21. doi:10.1109/MCG.2014.69. 2