A Creative First Assignment in the Modern Graphics Pipeline

Elodie Fourquet[1] and Lillian Pentecost[1,2]

[1]Colgate University, Hamilton, NY, USA
[2]Harvard University, Cambridge, MA, USA

Figure 1: *The Snail by Henri Matisse, 1953, Photo: © Tate, London 2018 (left), next to the student's sketch which plans the WebGL result (middle). The wireframe images (right) illustrate a functionality programmed to subdivide a curve in a fan of triangles (close-ups generated using fans of 10 and 20 triangles).*

Abstract

This paper describes a first assignment in an Introduction to Computer Graphics course taken by undergraduate students at a liberal arts college. The assignment marries the technical challenges found at the lowest level of the modern graphics pipeline with the artistic concerns of reproducing a piece of art. To do so, students extend provided code in WebGL, which includes GLSL shaders and no additional libraries, to reproduce a work of art of their own choosing. This task requires the students to involve themselves simultaneously in the most technical and most artistic aspects of computer graphics. Such an inter-disciplinary approach helps to reach a more diverse audience of computer graphics learners.

CCS Concepts

•***Social and professional topics*** → ***Computer science education;*** *Computational science and engineering education;* •***Computing methodologies*** → ***Graphics systems and interfaces;***

1. Introduction

In the last decade, academic and industry-based computer graphics research developed a modern version of the graphics pipeline that obtains high performance in 2D and 3D graphics by running shader instructions directly on the GPU. This "new" pipeline provides significantly more flexibility for implementers thanks to "shaders (also called programs) [which] replace the hard-wired transformation and lighting model of the GPU (also called fixed function pipeline) with a programmable one" [Bau07]. The novel architecture of the pipeline innovation creates an educational challenge.

The educational challenge is motivational. Students want to see pictures right away. However, minimal working *HelloGL* programs developed for the new pipeline are relatively long and complex, varying from 50 [mdn18] to 150 lines of code [Tho13], most lines a mystery to a student new to computer graphics. Merely reading them requires the student to combine inter-dependent modules in three languages (HTML, C-like and JavaScript), a significant challenge [RME14]. It is hard to see how the modules interact to put the image on the screen. Unsurprisingly, the more performant and flexible the graphics pipeline is the more complex the programming environment becomes [BWF17] and the more set-up is needed before a student sees any result. Abstracting away detail by providing high-level libraries has been proposed [FWW13, AB15, TRK17], as has been teaching computer graphics (CG) as a whole in a top-down rather than in a bottom-up fashion [AS14]. These approaches allow students to get results early, which stimulates their interest in the subject, but at the cost of hiding the open-ended power of modern graphics hardware, which is unattractive to technically-minded students.

This paper proposes an initial assignment that provides students with an aesthetically satisfying result while exposing them to low-level accelerated code from the start of the course. The code they write reproduces a 2D work of art of their choice, such that the output is attractive to them, and working in 2D gives them exposure to pipeline programming without the complexities of 3D geometry, which they learn later in the course. Indeed, the assignment takes advantage of the flexibility of the modern programmable pipeline. Students learn to model and visually debug the rendered output, which are skills that are easy to learn in 2D, and they carry these skills over to 3D later.

In this paper, we first present the assignment as seen by the students. Next, we discuss their results with comments on what features the students coded, how the assignment can be tuned, and common pitfalls to avoid.

2. Assignment Description

The first assignment in this Introduction to Computer Graphics course requires fourth-year undergraduate students to reproduce a work of art. In doing so, they learn how to manipulate the geometric primitives of WebGL 2.0 (the JavaScript API based on OpenGL ES 3.0 API), get familiar with the mechanisms that process primitives using the GPU vertex and fragment shaders, and gain experience coding low-level model data structures and graphics functions. Specifically, students focus on defining the 2D geometry in terms of triangles, associating each vertex with attributes and structuring images as collections of JavaScript objects.

When students reproduce a work of art, they must think about the graphical structure of the work and plan how to map it onto graphics primitives. This is done in three stages, which are described in Sections 2.1, 2.2, and 2.3. An example of this process is illustrated in Figure 1, showing the sketch prepared by a student and their WebGL code output. The learner gets both agency and ownership as they begin making design choices that affect the final result.

2.1. Work of Art Selection

Each student starts their assignment by selecting a 2D work of art. Students are asked to find their piece in a library art book, as opposed to an online photograph, because art historians situate the artist and the work and provide trustworthy representations of a work's coloration. The students are required to select a piece that contains sufficiently complex geometry in order to use several drawing modes in WebGL, and this is usually the student's first exposure to thinking in terms of graphics primitives.

The leftmost image of Figure 1 is the work of art selected by a student. Found in a book of cutouts and collages of Henri Matisse [EM78], *The Snail* has challenging geometry, containing both sharp and smooth edges. The vibrant colors and unique layout inspired the student, and this is a good example of a manageable challenge. Reproducing *The Snail* as a first WebGL implementation demonstrates the rewarding experience and goals students set for themselves in this assignment.

2.2. Sketching & Planning

After choosing a work of art, the student is required to hand-draw a sketch of it with clearly labeled coordinates that quantify its 2D features. Good geometric planning minimizes later trial and error editing of data structures.

The second image of Figure 1 is the student's tracing of *The Snail*. The basic structure of the piece, including the effect of layering the collage papers, is apparent in the sketch. The placements of the straight edges forming the individual geometric shapes are emphasized with coordinate labels. Other unique and complex features of the piece that increase its resemblance to the original are present but only suggested in the hand-drawn plan.

2.3. Coding

For this assignment, 150 lines of starter code, included in the supplementary materials, are provided. It comprises the repetitive code found in most graphics application [AH17], including

1. setting up canvas for rendering,
2. setting up data in the program,
3. creating two GLSL shader programs:
   - the vertex shader, which receives vertex position as 2-component vector and converts it to 4-component, together with its 4-component color attribute, and
   - the fragment shader, which copies each incoming interpolated color value to an outgoing fragment color,
4. creating buffer objects and loading data into them,
5. connecting data locations with shader variables, and
6. rendering.

Students come to understand this provided code as they make changes and additions to it. Specifically, they extend the code to introduce the data (item 2) that reproduces their work of art, organizing it into vertex buffer objects and loading the data with drawing mode in the WebGL API (item 4) to render primitives from array data. From this starting point, students understand the ubiquity of collections of triangles for high-performance graphics in a way that is unlikely to be apparent when specifying a sphere or loading human face mesh within a high-level graphics API.

Because their works of art have many, many triangles, students write JavaScript functions to create data. Those functions return arrays of 2D coordinates representing the geometry of their sketches. They also create JavaScript objects, grouping array data buffer, stride size and total size with the drawing primitive mode, to be loaded into WebGL [Par14, Chap 2.]. These JavaScript code abstractions make it easy to follow the stateful nature of the graphics context, thus maximizing performance.

Coding *The Snail*, the student made use of the stateful nature of the graphics context to draw each shape so that the overlapping effect was executed in accordance with the original piece, as shown in the middle image of Figure 1. First, the student implemented a simple version of the painting, determining precisely the normalized device coordinates for the rigid edges of each shape and their color. Then, the student wrote a JavaScript function to include the prominent curved features. Indeed, many students wrote functions that

discretized circles as triangle fans, based on a center, a radius and the number of triangles needed to achieve the desired smoothness. The right side images of Figure 1 illustrate the results of such a function, parametrized for an arc. Many students rendered in wireframe to compare the effects of using different number of triangles.

3. Results

This assignment was given to a class of twenty-four undergraduate students at a liberal arts college. Students had two weeks to complete their work. On its completion, all students could program the modern CG pipeline successfully, realising aesthetic goals with up-to-the-minute technology. Half of the students produced images of comparable quality to the ones included in this paper. Female students out performed male students: Figures 1, 2 and the middle two of Figure 3 were created by women. Next, we discuss features of the students' results and two pitfalls that reduced student learning.

3.1. Student Successes

We note two aspects of art selection that were important in developing students' programming ability and confidence. Students learned more when they chose a work that demanded careful geometric planning and had sufficiently complex geometry to necessitate procedural abstraction when editing their shapes in code. Successful students automated repetitive modeling tasks, making evident the connection between image structure and program structure.

The first aspect is selecting a challenging work of art. Figure 2 shows the result of a challenging selection, similar in quality to Figure 1. The freedom to choose provokes most students into ambitious choices. They enjoy the opportunity to explore further a work they admire, know, or may have seen. The *Three Musicians* painting by Picasso was selected by a student who had a personal attachment to the piece, having seen it multiple times in the Museum of Modern Art. The sketch demonstrates a carefully thought out plan that guided its recreation in WebGL.

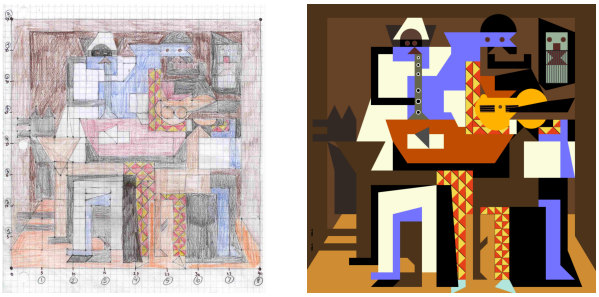

Figure 2: *Sketch based on the Three Musicians, a painting by Pablo Picasso (1921), and the WebGL created image.*

The student's sketch retains the abstract geometry of the original work, capturing occlusion by layered drawing. Some details omitted from the sketch appear in the final WebGL image. Among them are holes on the flute, implemented by overlapping circles, and the patterned lines in the friar's veil and guitar strings, and these details make each object more salient. These additions indicate the increased creative opportunities that occur as the graphics environment is mastered.

Thus, the final interpretation shows the student incrementally adding features in code to improve the aesthetic of the WebGL image. For example, the repetition of triangles for each paw of the dog, a subtle detail simple to code, visually separates the animal legs from the background wall. Some complexities that are modifications of the original work are sometimes added in student designs. The harlequin's costume, for example, has on the right leg a pattern that is a reinterpretation created in the sketch by the student.

A second approach to selecting a piece is to choose a highly geometric work, sufficiently complex because it is rich in shapes and patterns. Works by Burgoyne Diller, Bryce Hudson, and Wael Shawky shown in Figure 3 were used, and other well-known works were the paintings of Robert Delaunay, Frank Stella and Kazimir Malevich. These strongly geometric works contain repetitions, smooth shapes, and sophisticated layering. They require students to master data abstraction, looping, and functional decomposition to produce program structure that reproduces image structure.

Figure 3 shows more examples of images that challenged students to innovate in code. *Composition 1938* (left images of Figure 3) requires code that renders open circles. A first try might render two overlapping circles, the second using the background color, but the blue lines behind preclude this approach. Instead, this student wrote a function that generates a circular band using a triangle strip. *Drawing 58* (right images of Figure 3) consists of two areas separated by an oblique divide. Each area is tiled by lozenges, yellow in lower area, grey in the upper. Each area is easily constructed functionally, but the dividing lines requires a special case computation of vertex positions such that the student's code reproduces the sharp divide of the composition. In the second set of images in Figure 3, *Untitled Composition (#43)*, careful ordering of primitive definitions facilitates editing the geometry. Indeed, the student recreating it used primitive overlappings to give the 3D effects of the viewed composition and wrote code to generate elliptical geometry with triangle fans. Finally, to effectively reproduce the third set of images in Figure 3, *Third Theme*, the student organized collections of rectangles in separate lists according to color attribute.

3.2. Common Pitfalls

Two common pitfalls reduce student learning. Some students select artworks that are overly simple. If the piece lacks geometrical challenge, students get little exposure to the programming environment. For example, six students selected purely geometric works by Piet Mondrian. Because a small number of primitives approximates the geometric layout, most of these students omitted drawing a careful sketch. The generated images were mere imitations of the originals and the students learned little. This pitfall demonstrates the importance of instructor supervision, feedback and guidance in the selection of works of art.

The second pitfall is more complex. It is important that abstractions in the code reinforce abstractions in the image, and vice versa. A weak image abstraction in a poorly thought out sketch is shown in the left image of Figure 4. Having insufficiently planned the geometry of the main elements, the most important being the rooster,

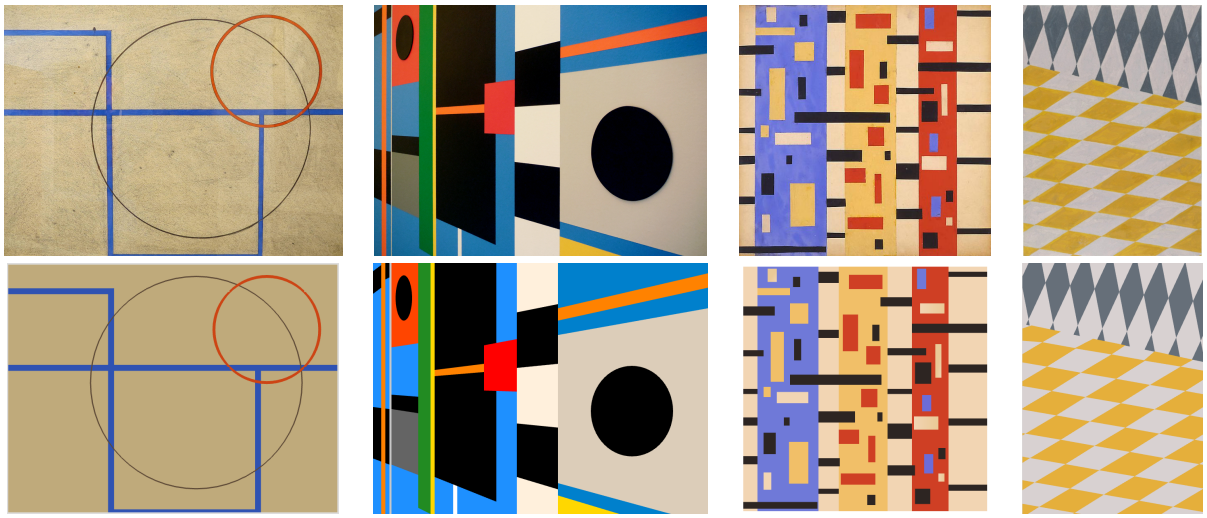

Figure 3: *Original works (top) and students' WebGL reproductions (bottom) from left to right: Composition 1938 by Burgoyne Diller (1934), Untitled Composition (#43) by Bryce Hudson (2014), Third Theme also by Diller (1939), and Drawing 58 by Wael Shawky (2010).*

the student wasted time experimenting with gradient in shaders. As a result, the generated image is poor, the rocks being insufficiently salient; the gradient simulating the shaded form is poorly executed and doesn't compensate for the inadequate abstraction of the geometry.

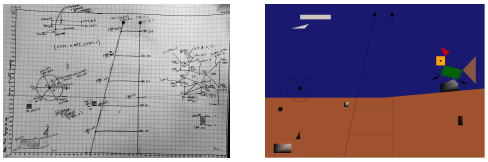

Figure 4: *Sketch based on the Landscape with Rooster painting by Juan Miró (1927) and the generated WebGL image.*

During the next offering of the course, mid-progress feedback will be provided twice during the assignment: first, a discussion of abstraction in the sketch and second, a code review as they write their WebGL program. We observe that students who worked incrementally, refining and elaborating abstractions as they went along, learned the most in completing the assignment. The goal is to have students learn to abstract the structure of scenes and images as a guide to abstracting the structure of code, and close and personal attention to each student's work is required to achieve this.

3.3. Conclusion

This paper presents a first CG assignment for undergraduate students that places them in the position of both the programmer and the artist. Taking this approach throughout the course helps them develop better software and makes them more well-rounded individuals. The traditional divide between art and technology can be overcome in our computer graphics classrooms [CMA10], and this assignment is an effort to innovate across that divide.

References

[AB15] ACKERMANN P., BACH T.: Redesign of an Introductory Computer Graphics Course. In *EG 2015 - Education Papers* (2015), Bronstein M., Teschner M., (Eds.), The Eurographics Association. 1

[AH17] ANGEL E., HAINES E.: An Interactive Introduction to WebGL and Three.JS. In *ACM SIGGRAPH 2017 Courses* (New York, NY, USA, 2017), SIGGRAPH '17, ACM, pp. 17:1–17:95. 2

[AS14] ANGEL E., SHREINER D.: *Interactive Computer Graphics with WebGL*, 7th ed. Addison-Wesley Professional, 2014. 1

[Bau07] BAUCHINGER M.: *Designing a Modern Rendering Engine*. PhD thesis, TU Wien, Vienna, Austria, 2007. 1

[BWF17] BALREIRA D. G., WALTER M., FELLNER D. W.: What we are teaching in Introduction to Computer Graphics. In *EG 2017 - Education Papers* (2017), Bourdin J.-J., Shesh A., (Eds.), The Eurographics Association. 1

[CMA10] COMNINOS P., MCLOUGHLIN L., ANDERSON E. F.: Selected Papers from the SIGGRAPH Asia Education Program: Educating Technophile Artists and Artophile Technologists: A Successful Experiment in Higher Education. *Computer Graphics 34*, 6 (2010), 780–790. 4

[EM78] ELDERFIELD J., MATISSE H.: *The cut-outs of Henri Matisse*. G. Braziller New York, 1978. 2

[FWW13] FINK H., WEBER T., WIMMER M.: Teaching a Modern Graphics Pipeline Using a Shader-based Software Renderer. *Computers & Graphics 37*, 1 (2013), 12–20. 1

[mdn18] Hello GLSL. https://developer.mozilla.org/en-US/docs/Learn/WebGL/By_example/Hello_GLSL (accessed January 2018), 2018. 1

[Par14] PARISI T.: *Programming 3D Applications with HTML5 and WebGL*. O'Reilly Media, Inc., 2014. 2

[RME14] REINA G., MÜLLER T., ERTL T.: Incorporating Modern OpenGL into Computer Graphics Education. *IEEE Computer Graphics and Applications 34*, 4 (2014), 16–21. 1

[Tho13] THOMAS G.: WebGL Lesson 1–A triangle and a square. https://github.com/gpjt/webgl-lessons/blob/master/lesson01/index.html, 2013. 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