On the Pedagogy of Teaching Introductory Computer Graphics without Rendering APIs

Minsi Chen [1], Zhijie Xu [1] and Wayne Rippin[2]

[1]Department of Computer Science, University of Huddersfield, UK
[2] Department of Electronics, Computing and Mathematics, University of Derby, UK

Abstract

Teaching modern computer graphics programming has become increasingly challenging due to the advancement in the granularity of application programming interfaces (APIs). In this paper, we put forward a discussion on the pedagogical value of implementing a software rasteriser prior to tackling the issues of learning modern graphics APIs and shader programming. An API-free approach to teaching introductory computer graphics along with its assessment strategy are presented. Our observation found that students were more effective and confident in learning and using modern rendering APIs when subsequently studying advanced real-time graphics.

CCS Concepts

•***Social and professional topics*** → *Model curricula; Student assessment;* •***Computing methodologies*** → *Rasterization; Ray tracing;*

1. Introduction

Computer graphics (CG) from its introductory form to more advanced topics are embedded in many computer science curricula either as an elective or as a core component. This field consists of both theoretical elements stemmed from mathematics and computer science, as well as practical applications that are appropriate for a much wider spectrum of audiences [McC06].

The theoretical treatment of CG has been comprehensively written in well-known textbooks such as Foley *et. al.* [FvDFH90] and Shirley *et. al.* [SM09]. In addition to this, there are also numerous widely adopted textbooks that are closely coupled with common rendering APIs such as OpenGL, see examples [AS11a, SWH15, HBC10].

Introductory CG topics commonly consist of raster-level concepts and algorithms that underpin the fundamental working of modern real-time rendering pipeline and the design of rendering APIs. From a pedagogical perspective, the necessity of assimilating these low-level processes as a requisite to further progress onto more advanced topics in this area has been actively debated.

As other scientific subjects, we do hold the belief that these fundamental concepts are necessary to form a structured approach to progressively studying CG. However, such low-level details are often hidden when graphics processing units (GPUs) and rendering APIs are adopted for practical exercises and assignments. Thus, students often found it less relevant to work through the nuances of the raster graphics pipeline and the core mathematics.

As modern rendering APIs become increasingly shader centric and hardware resource management oriented, we perhaps put an unbalanced focus on elucidating the working of an API instead of the enabling theories and techniques. In an introductory CG module, API-led teaching and assessment may not facilitate an effective platform for students to combine theories with practice.

In this paper, we present a portfolio of assignment design that tasked students to implement raster-level algorithms and shading techniques. The method proposed were primarily developed for science and engineering students who require a balance of theoretical and applied treatment of the introductory elements of CG.

2. Related Work

The importance of raster-level algorithms in teaching introductory CG was debated by Angel *et. al.* [ACSS06]. Peter Shirley, one of the author, argued that implementing your own raster-level algorithms could lead to more commanding understanding of the working principle of rendering pipeline. This in turn also enabled students more learn and use rendering APIs more effectively.

More recently, the prevalence of programmable shaders has instigated a re-examination of the level of details required for an introductory CG course. As discussed in [AS11b, RME14], a shader centric approach to teaching CG could indeed afford opportunities for students to practice relevant mathematical concepts. Since transformation and lighting are no longer performed as an opaque process, students are required to apply those key computation in forms of shader programming statements.

The design of bottom-up methods for teaching low-level algorithms and pipeline processes were discussed in [SBG10,FWW13]. Both papers presented a software based renderer as a sandbox in which students could interact with and experiment with fundamental rasterisation algorithms. Kolingerova proposed a rationale for integrating applied computational geometry to CG education [Kol08]. Through an analysis of several core topics in computational geometry, the author viewed the integration as a means of developing students' abstract as well as algorithmic thinking.

Educational practitioners often advocate the importance of linking students' prior experiences to the teaching new concepts and theories. Taxén discussed a constructivist's approach to teaching introductory CG in [Tax04]. This was, in effect, a top-down approach whereby students were first exposed to the results before further exploring the constitutive solutions.

In the remainder of this paper, we will detail an API-free approach to teaching introductory CG and demonstrate its results using works produced by our students.

3. Methods

At the University of Derby, the subject of CG is taken as a core component by students from two degree programmes - *BSc Computer Science* and *BSc Computer Games Programming*. Since its redesign in 2012, the topic has been covered in the following two progressively structured modules:

- Graphics I: Introduction to the fundamentals of raster pipeline
- Graphics II: Real-time CG based on modern APIs

Quantitatively, circa. 50% of the *Graphics I* cohort (average 95-110 students) study further topics in CG focusing on real-time graphical effects.

3.1. Module Contents

In our introductory CG Graphics I module, students were presented with the top-level view of the raster graphics pipeline (as depicted in Figure 1) in the first lecture. This created the pillar upon which the core concepts and algorithms of raster graphics were built and structured.

Figure 1: *A simplified top-level view of the raster graphics pipeline.*

The core contents listed below were delivered over 11 weeks in a 12 week semester.

- Introductory CG (Duration: 8 Weeks)
  - Primitive representations
  - Line rasterisation
  - Scanline conversion and clipping
  - Geometrical transformation and viewing in 3D
  - Pixel operations including z-buffering, alpha blending and stenciling
  - Whitted's Ray-tracing method
  - Basic shading up to Phong and Blinn-Phong model
- Mathematics (Duration: 2 Weeks)
  - Vectors and related operations in R^2 and R^3
  - Matrices for describing transformations in R^2 and R^3
  - Implicit and explicit functions for geometry
- Refresher C++ Programming (Duration: 1 Weeks)
  - Representing core raster graphics components using OO
  - Use of fundamental data structures

As a prerequisite, students had already encountered the basic elements of OO programming in C++. The mathematical elements required for representing geometrical primitives and raster operations were introduced *in-situ*. Students spent on average four hours a week in lectures and labs with tutor supervision.

3.2. The Design of an API-free Assignment

The core design principle of our assignment was to facilitate the application of fundamental raster-level concepts and algorithms in practice. We tasked the students to build a software rasteriser without using any functionality from a rendering API.

In the assignment brief, exam style tasks were posed to the students. We further structured them into *basic* and *advanced* categories to help students identify the intended progression of these tasks as shown in Table 1. The basic tasks embedded the threshold knowledge where all students should command as part of the learning objectives. Advanced tasks primarily concerned with differentiating students' ability to assimilate the deeper aspect of raster graphics.

	Tasks
Basic	Drawing 2D line segments of an arbitrary slope
	Filling arbitrary 2D polygons in scanline order
	Ray-primitive intersection tests
	Model-view-projection transformation
Advanced	Interpolated shading of lines and polygons
	Visibility testing and clipping
	Drawing overlapping transparent polygons
	Applying basic illumination model

Table 1: *The grade distribution of API-free and API-led assessments.*

3.3. API-free Starting Framework

The process of building a software rasteriser from scratch can be overwhelming to many undergraduate students. In order to alleviate this, we adopted the Test Driven Development methodology (TDD) by providing the skeleton of a basic rasteriser framework along with a number of unit test cases to the students.

The starting framework, appropriately named `TinyRaster`, was written in C++ as a Windows GUI application. The core

components of a raster pipeline were encapsulated in a collection classes as depicted in Figure 2. The *Rasterizer* class encapsulated the key attributes and operations of a 2D raster pipeline. Additionally, vector maths utilities were provided to the students.

Each task listed in Table 1 has a corresponding unimplemented method in the framework as shown in Listing 1.

```
void Rasterizer::ScanlineFillPolygon2D(const Vertex2d * vertices,
     int count)
{
  //TODO:
  //Q 2.2 Implement the Rasterizer::ScanlineFillPolygon2D method
       so that it is capable of drawing a solidly filled polygon.
  //Use Test 4 (Press F4) to test your solution, this is a simple
       test case as all polygons are convex.
  //Use Test 5 (Press F5) to test your solution, this is a complex
       test case with one non-convex polygon.
}

void Rasterizer::ScanlineInterpolatedFillPolygon2D(const Vertex2d
     * vertices, int count)
{
  //TODO:
  //Q. 2.4 Implement Rasterizer::ScanlineInterpolatedFillPolygon2D
       method so that it is capable of performing interpolated
       filling.
  //Use Test 7 to test your solution
}
```

Listing 1: *A snippet of the Rasterizer class containing unimplemented methods for filling 2D polygons.*

Furthermore, we constructed unit test cases related to the tasks posed in the assignment brief to evaluate the correctness and completeness of the solution given by a student. Listing 2 shows an example test harness for evaluating the implementation of the scanline conversion of 2D polygons using interpolated filling.

```
void AssignmentTest07(Rasterizer * rasterizer)
{
  rasterizer->SetGeometryMode(Rasterizer::POLYGON);
  rasterizer->SetFillMode(Rasterizer::INTERPOLATED_FILLED);

  rasterizer->ScanlineInterpolatedFillPolygon2D(grad_rectangle, 4)
      ;
  rasterizer->ScanlineInterpolatedFillPolygon2D(grad_triangle, 3);
  rasterizer->ScanlineInterpolatedFillPolygon2D(grad_square, 4);
  rasterizer->ScanlineInterpolatedFillPolygon2D(grad_pentagon, 5);
}
```

Listing 2: *An example test case for testing the interpolated 2D polygon shading method.*

3.4. Assignment Distribution and Assessment

The starting framework and assignment instruction were distributed through our Blackboard-based web portal. Students were given the complete set of assignment tasks after four weeks of teaching, thus giving them an average one week to work on each task.

Similar to traditional exams, our intention was to objectively evaluate the students' understanding of the theoretical aspect of the subjects. Each test case was assigned a maximally attainable grade that was directly correlated to the difficulty level of a task. Solutions given by the students were graded based on *correctness and completeness*, *robustness and efficiency*, and *coding style*. For examples, interpolated filling was considered more challenging than filling polygons with a solid colour; scanline conversion was deemed more robust and efficient solution compared to flood fill.

4. Results

The API-free assignment was used as a summative assessment component in 2012-2013 and 2016-2017 academic years. The class size was 95 and 110 respectively. A pass grade D was awarded to a student if correct and complete solutions were provided to all basic tasks. This indicated that the student had met the threshold learning objectives. Higher grades were awarded based on the solutions provided to the advanced tasks. Based on this criteria, the success rate was respectively 84% (2012-2013) and 88% (2016-2017), see Table 2.

	API-free		API-led
	Yr.12-13	Yr.16-17	
Pass Rate	84%	88%	83%
Grade A	10%	12%	6%
Grade B	41%	55%	20%
Grade C	25%	12%	41%
Grade D	8%	9%	16%

Table 2: *The grade distribution of API-free and API-led assessments.*

Around 90% the students who passed the API-free assessment also attempted a varying combinations of advanced tasks. In particular, interpolated filling, application of basic illumination model and blending of transparent polygons were most received solutions. A grade C was awarded to submitted works included flawed and partial attempts on the advanced tasks. Students given grade B and above demonstrated working solutions to all tasks specified in the assignment.

The last column shows the average pass rate and grade distribution between 2013-2016 when the assignment was application focused and based on the use of OpenGL 4.3 API. The size of the cohort in these academic years was similar to those in the API-free periods. Whilst the averaged pass rate was statistically similar between API-free and API-led, the performance on the latter was skewed towards the low end of the grade scale.

5. Discussion

In the API-free assignment, the majority of students found 2D raster operations intuitive to understand and largely straightforward to implement given the starting framework. The noticeable obstacles were the application of linear interpolation and the handling of vertices when filling non-convex polygons.

The transition into 3D rendering was noticeably more challenging as students first encountered the mathematical concepts related to spatial and projective transformations. From our experience, introducing 3D rendering through ray tracing appeared more easily applicable to most students compared to the model-world-view-projection transformation pipeline. The notion of ray tracing was more easily associated to the physical analogy they experienced and observed in the past.

In the case of API-led assignment, only around 25% of the students were able to attain Grade B and above by completing advanced tasks such as applying basic illumination models. This was

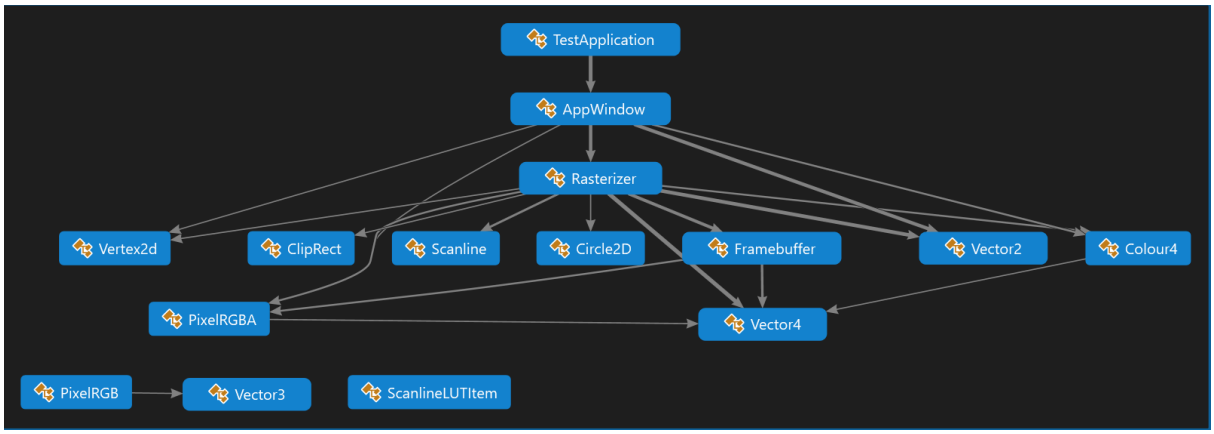

Figure 2: *The core components of* `TinyRaster` *consist of geometry primitives, vector maths utilities, framebuffer and the rasteriser.*

reflected in the finding of our module surveys where 82% students considered the API-led assessments disconnected from the teaching of core raster graphics concepts. From our perspective thought, the parallel introduction of the fundamentals and API may have also distracted the students' focus on the intended learning objectives.

Over the course of 5 years, we found, albeit empirically, that students with experiences in implementing raster-level algorithms were more proficient at self-studying modern rendering APIs. In our 2016 qualitative module survey, 91% of the students found the API-free assessment challenging but rewarding given the opportunities to work on the low-level aspects of rendering pipeline.

Despite the combination of qualitative module surveys and grade attainment distribution, our current evaluation lacked an objective measurement of the transferability of raster level knowledge to future API learning. It is possible, however, to acquire the relevant data by using a standardised written test in which students are asked to discuss the design of rendering API in relation to the fundamental raster pipeline.

6. Conclusion and Future Work

The merit and benefit of an API-free approach to teaching introductory graphics may not be immediately evident solely on the outcome of assessment. However, the hands-on experience in implementing raster-level algorithms can enable students to more deeply assimilate the abstract concepts underpinning the rendering pipeline. We believe this will markedly benefit them when they take on a proactive and autonomous approach to learning and applying modern graphics APIs.

In the current academic year 2017-2018, we are using the 2D rasteriser assignment in the first year undergraduate programming module to a group of Computer Games Programming students. We hope to examine the feasibility of introducing certain elementary CG concepts early in an undergraduate computer science curriculum.

References

[ACSS06] ANGEL E., CUNNINGHAM S., SHIRLEY P., SUNG K.: Teaching computer graphics without raster-level algorithms. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education - SIGCSE '06* (2006), p. 266. 1

[AS11a] ANGEL E., SHREINER D.: *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL*, 6th ed. Addison-Wesley Publishing Company, USA, 2011. 1

[AS11b] ANGEL E., SHREINER D.: Teaching a shader-based introduction to computer graphics. *IEEE Computer Graphics and Applications 31*, 2 (2011), 9–13. 1

[FvDFH90] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F.: *Computer Graphics: Principles and Practice (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. 1

[FWW13] FINK H., WEBER T., WIMMER M.: Teaching a modern graphics pipeline using a shader-based software renderer. *Computers & Graphics 37*, 1–2 (Feb. 2013), 12–20. 2

[HBC10] HEARN D. D., BAKER M. P., CARITHERS W.: *Computer Graphics with Open GL*, 4th ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2010. 1

[Kol08] KOLINGEROVÀ I.: Computational geometry education for computer graphics students. *Computer Graphics Forum 27*, 6 (2008), 1531–1538. 2

[McC06] MCCRACKEN C. R.: Issues in computer graphics education. In *ACM SIGGRAPH 2006 Educators program on - SIGGRAPH '06* (2006), p. 29. 1

[RME14] REINA G., MÃIJLLER T., ERTL T.: Incorporating modern opengl into computer graphics education. *IEEE Computer Graphics and Applications 34*, 4 (July 2014), 16–21. 1

[SBG10] SCHWEITZER D., BOLENG J., GRAHAM P.: Teaching introductory computer graphics with the processing language. *J. Comput. Sci. Coll. 26*, 2 (Dec. 2010), 73–79. 2

[SM09] SHIRLEY P., MARSCHNER S.: *Fundamentals of Computer Graphics*, 3rd ed. A. K. Peters, Ltd., Natick, MA, USA, 2009. 1

[SWH15] SELLERS G., WRIGHT R. S., HAEMEL N.: *OpenGL Superbible: Comprehensive Tutorial and Reference*, 7th ed. Addison-Wesley Professional, 2015. 1

[Tax04] TAXÉN G.: Teaching computer graphics constructively. In *Computers and Graphics (Pergamon)* (2004), vol. 28, pp. 393–399. 2