Teaching Graphics To Students Struggling in Math: An Experience

Amit Shesh[†]

Illinois State University, Normal IL USA

Abstract

Undergraduate students with a negative attitude towards Math present a unique challenge when teaching computer graphics. Most meaningful concepts in computer graphics involve directly working with Math in the classroom, and implementing tasks in programs requires a reasonable grounding in Math concepts and how to apply them. This paper presents a semester-long experience in using three strategies to address difficulties faced by computer science students who are interested in learning computer graphics, but feel less confident or uninterested in Math. Similar to how Math is taught in schools, we focus on giving students more and more practice in implementing progressively complex visual tasks. Students accomplish some tasks individually to develop a basic understanding before completing other tasks in groups. Students achieve more in a semester than before, and our preliminary observations show a higher rate of completion by students, moderate gains in performance in individual assignments and significant gains in overall class performance.

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

1. Introduction

Although all computer science (CS) degree programs have supporting Math requirements, there are relatively few subjects in the undergraduate computer science curriculum that directly and significantly use Math. Computer graphics (CG) is one such *mathematically intensive* subject, generally offered as an upper-level (fairly popular) elective. Teaching computer graphics generally involves a healthy dose of reviewing Math and connecting it to graphics-specific tasks such as modeling, animation, etc. Conversely, being familiar with not only the basic mathematical concepts but also their application is critical for a student to truly understand computer graphics and succeed in such a course. This presents unique challenges for the students and the instructor.

In our undergraduate CG course, we regularly encounter students who are *wary* about Math in general. When working out a concept that involves Math, students are often unable to recall concepts learned in high school geometry or even in a recent calculus course. In other cases when they recall the concepts correctly, they are not able to connect them to the practical graphics task at hand (e.g. how to find the angle and axis of rotation to align one cylinder to another). We have observed students showing this behavior and attitude irrespective of their grade and overall GPA and despite having taken several Math courses as supporting requirements. As a result, students often do not demonstrate thorough understanding (as evidenced by their performance in exams), or are unable to accomplish as expected (submitting incomplete assignments, later recalling how they took an inordinate amount of time on a particular part). Many students often informally convey how they "worked harder than they thought" or "did not feel prepared for all the Math". This paper describes strategies for teaching and assigning course work that we attempted, which resulted in students accomplishing more tasks within the same time frame, with significantly better overall performance.

Despite experiencing them in a medium-sized college in the US, we feel the challenges we face are shared with varying degrees by most computer graphics instructors. The extent and nature of supporting Math requirements for a CS program may have an effect on the severity of this prob-

[†] ashesh@ilstu.edu

DOI: 10.2312/eged.20151023

lem. These requirements are influenced by various factors: whether the program is housed within engineering or math departments, time-to-degree and credit-hour requirements, educational objectives of the department, etc. [ACM14]. The curriculum of an undergraduate CG course typically requires knowledge of linear algebra (vectors, matrices, etc.), geometry and trigonometry, not all of which may be part of college-level Math courses taken earlier. Another possible reason may be the time gap (often significant) between when students typically take Math courses and the CG course. For some concepts that students may not have seen since high school, the effects of this time gap may be more detrimental. A third reason may be the way in which Math courses are taught. Often these courses are taught in other departments that emphasize more on theory, which leads to an impedance mismatch with the applied nature of computer science and specifically graphics (e.g. students know the Math, but struggle with how to implement or use it within a computer program). Our degree program, like many others, have elements of all these factors which leads us to believe our positive experience can be helpful to other instructors.

In the content of a typical undergraduate CG course, Math is used in two ways. First, math can be used directly to solve visual problems (for example, creating triangulated meshes of mathematical solids, computing ray-object intersections, etc.). Secondly, mathematical constructs and operators can be used as-is to achieve a visual goal (for example, implementing through matrix transformations a revolving light on top of a police car that moves with it, etc.). Thus a successful graphics programmer at this level does not have to be an expert in Math, but simply know how to correctly *identify* which Math concept is relevant to the task at hand, and then *correctly apply* it. Indeed a large number of available libraries already implement the nitty-gritty of the Math: identifying and learning how to use them to complete a visual task should be the primary objective of a student. We attempt to achieve this by making students complete *more* and *progressively complex* visual tasks that force them to apply and combine relevant Math concepts.

The two main challenges in giving more practice to students are the constraint of time, and the difficulty of designing assigned course work that covers the Math, achieves the desired complexity of tasks and is still practical for most students to complete them without losing motivation. We tried three specific ideas. First, we included modeling and rendering tasks that asked them to directly implement Math derived in class or used to create existing models. Secondly, we built a more thorough understanding of transformations, matrix algebra and coordinate geometry by assigning specific visual tasks of increasing complexity. Thirdly, we combined individual and group work in a way that allowed students to practice individually before working together to combine programs and complete tasks that benefited from working in groups. Our experience shows how our students (1) completed the individual tasks well and scored higher on comparable tasks than previous semesters (2) achieved more tasks in a group than individually possible in the same time frames (3) performed slightly better in individual assignments and significantly better overall in the course.

2. Related Work

We did not find much research done on addressing the particular problem that we face in our course: teaching a foundational CG course to students with negative attitudes towards math. However many innovative approaches to teaching computer graphics that have risen over the past decades are relevant to this specific problem. Over the years, two ways of teaching a CG course have emerged: bottom-up (start with low-level operations and building upon them, use low-level APIs like OpenGL), and top-down (focussed on high-level problem solving and assembling applications, use high-level API) [SS03, ACSS06, TBN06]. Although the bottom-up approach mimics most computer graphics textbooks and is often the choice for a foundational treatment of the subject, it requires an intense mathematical foundation [AK07]. Top-down approaches ameliorate this drawback somewhat by making the course application-focussed instead of concept-focussed: examples include using applications such as game design [HS05, AP09], top-down API-agnostic [SBG10] or specific APIs [Owe92]. The objective of our course is to introduce undergraduate computer science students with no prior CG background to the conceptual basics of computer graphics and what it takes to produce and render a 3D scene. Students subsequently enroll in our game programming course, apply graphics in other contexts or in some cases, consider graduate study. Thus we feel a bottom-up approach is best suited for our course.

3. Our approach

3.1. Background of course

Our CG course is the only course in computer graphics offered in the degree program. It is offered as an upper-level elective. The pre-requisite course covers data structures and algorithms using C++. All students in this course have had about 1.5 semesters of C++ programming from earlier courses in our own program before they enter this course (one student in Fall 2014 entered this course without earlier courses in C++ in our program, but did not face any problems with the language). All students, through indirect pre-requisites (i.e. courses required to take a pre-requisite for this course), take the same 2 calculus courses and at least 1 discrete math course *before* they enter this graphics course. A course in linear algebra is not required.

This paper presents data pertaining to this course from 4 semesters. The same instructor (the author) taught this course during all 4 semesters without teaching assistance. The number of students during these semesters were 12, 21,

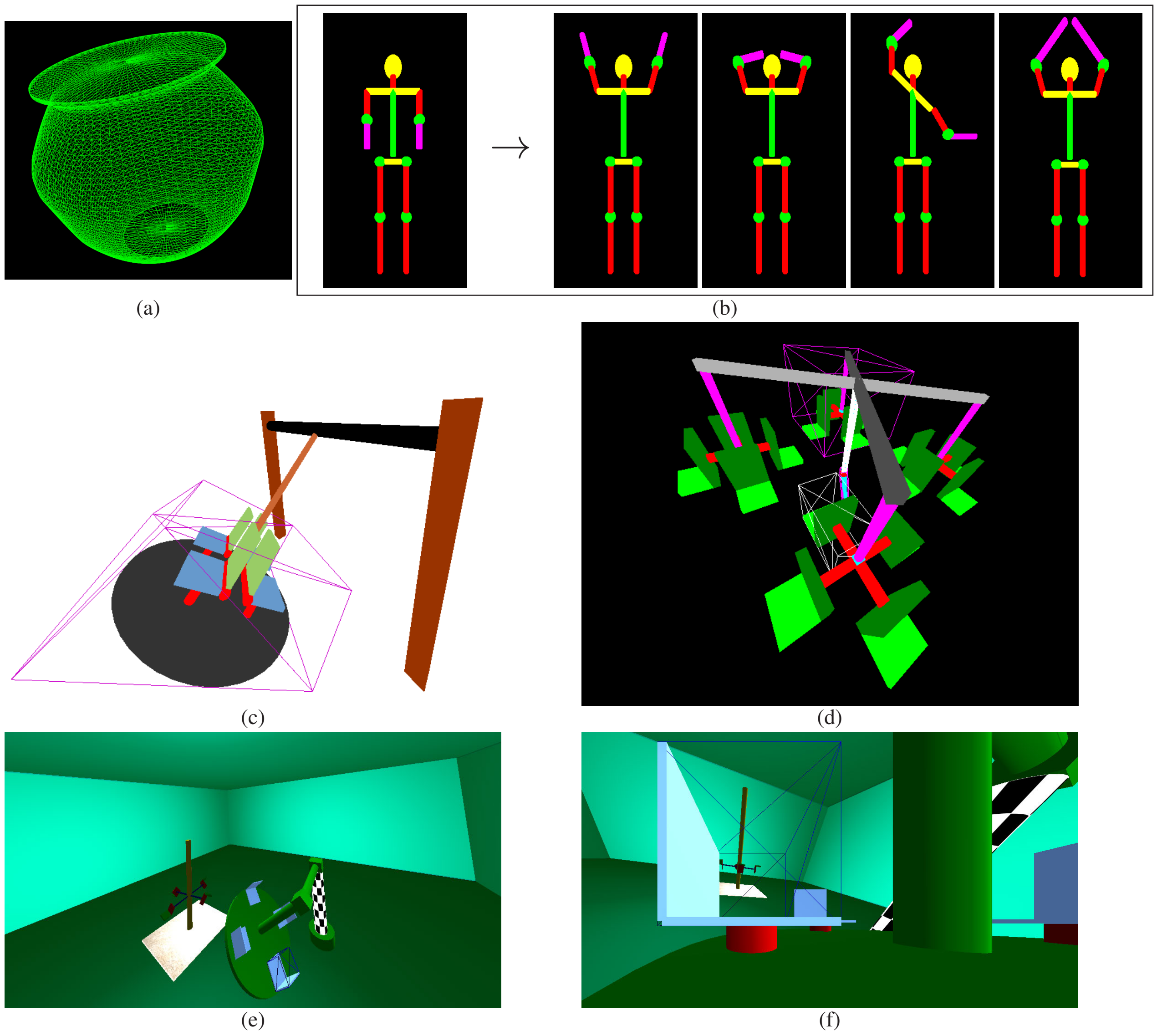


Figure 1: *Example results from student programs. (a) Surface of revolution created by a student. (b) Given a humanoid model in scene graph form, students were asked to modify the model directly to produce various poses of the "Y", "M", "C", "A" dance [ymc14]. (c-d) An amusement ride constructed by two students individually (satisfying constraints from Task 3-2 in Section 3.3). They also created bounding boxes per node, marked in the model file and drawn in wire frame boxes (see Task 3-3 in Section 3.3). (e) Two students combined their rides into one scene with moving lights and textures (Task 4-1 in Section 3.3). The part shown in the box is the focus of the moving camera (see Task 4-2 in Section 3.3). (f) A view from the moving camera pointed at the part illustrated in the box in (e).*

12 and 22 respectively. This size and trend is fairly typical for this course.

The course typically covers composite and hierarchical transformations, modeling, lighting, texture mapping, shadow techniques and ray tracing, roughly arranged in a canonical format (like [She13]). The course typically assigns 6 programming assignments. Each assignment is to be completed individually, typically in 2 weeks. In order, these assignments cover the following topics: 2D graphics, basic 3D modeling and transformations, hierarchical modeling and animation using scene graphs, lighting and texture mapping using scene graphs and a ray tracer. Some of the assignments are progressive, i.e. they build on previous assignments and thus our overall set of assignments follows the “four I-s” approach [PA14]. All assignments are in C++, use a suitable windowing library and typically use the GLM library [glm14] for matrix operations. We have adopted the shader-first approach since the last two years [Ang11, AS11] and use OpenGL 4.0.

We provide an overview of relevant math as various topics are discussed in the course. For example, coordinate geometry (cartesian and polar) are discussed early on, basic matrix operations like multiplication and inversion are discussed before introducing transformations and repeatedly reviewed throughout the semester. Mathematical solids and their analytical expressions are discussed during modeling and ray tracing, and vector algebra (i.e. vector math, products, etc.) are discussed before lighting and ray tracing. Students are allowed to bring a “cheat sheet” during exams so that they have an opportunity to write down the relevant Math formulas and expressions. We now motivate and discuss the three ideas mentioned earlier.

3.2. Application of analytical math

Basic trigonometry and coordinate geometry are used in almost every task in computer graphics. For example, when we introduce 3D models as triangular meshes, we typically start from simple objects like spheres. We explain how its equation in polar coordinates allows us to discretize it into triangles at a sufficient resolution.

Although students typically understand these concepts easily, they appear diffident when applying similar procedures elsewhere. In a phenomenon that is difficult to explain, we encounter with unfortunate regularity each semester students that appear phobic to anything that reminds them of their calculus class. This is evident in answers to exam questions and performance in assignments that asks them to work out the math before implementing it. A classic example is to determine the normals of a mathematical shape using partial differentiation in the process of creating its triangle mesh.

We ameliorate this situation by providing several opportunities for directly applying math, many of them as extra credit. For example in Fall 2014, students were asked to create a closed surface of revolution that can be physically manufactured (see Figure 1(a)). Later, students determine texture coordinates for spheres and boxes while attempting a ray tracer, and optionally determine ray-cylinder, ray-cone and ray-triangle intersections for extra credit. All of these exercises require the student to work out math and trigonometry and implement them in a program. While it may be argued that such modeling using direct math may be “advanced” for undergraduates, we feel these exercises give students an “end-to-end” experience of starting with Math and implementing it to produce and render a 3D scene. Several exam questions probe their understanding of how and when to apply Math.

3.3. Modeling and Manipulation

The overall task of creating a desired model and animating it in a specific way requires students to combine several skills and concepts. An example task may be to model an automotive that moves in a believable manner along a predefined path with headlights that move with it. To complete such tasks, the programmer must think of model assembly, desired animation and lighting effects in terms of specific transformations or change in coordinate systems. From our experience, exploiting this duality of transformations to produce a given effect is difficult to master for many students, and requires extensive practice. Our approach is to provide just that, with many tasks that ultimately lead to inserting specific transformations in a hierarchical model.

We start by providing students with a simple scene graph system implemented by us. This reduces the potential learning curve compared to mature and powerful libraries such as Open Scene Graph [ope11]. We then provide a simple scene graph model in XML to students and ask them to create “poses” by modifying only the model file. Next, we ask students to create a model with some structural constraints and produce specific animations. Lastly, we ask students to complete several tasks that do not require adding transformations, but focus on using them.

For example in Fall 2014 we gave students the following tasks, spanning across the 3^{rd} and 4^{th} assignments:

Task 3-1: Modify given scene graph: Use the provided model of a humanoid and create the ‘Y’, ‘M’, ‘C’,‘A’ poses from the famous song [ymc14] by modifying the model file only (see Figure 1(b)). This requires them to insert additional nodes in the model that gives desired degrees of freedom.

Task 3-2: Create and animate model: Create an amusement ride model using boxes, spheres, cones and cylinders with the following constraints (1) there should be at least 4 seats (2) seats should be attached to a main assembly (3) the model should be animated such that the seats should move in at least two ways (e.g. seats rotating about an axis that is itself revolving around another axis).

This task expects students to understand the iterative process of model assembly and scene graph modification to create the structure and enable it to move in certain ways. Figure 1(c-d) show some student examples.

Task 3-3: Bounding boxes: Calculate bounding boxes for each node of the scene graph that are aligned to the axes of their own coordinate system. These bounding boxes should be enabled and drawn per node as the model is animating.

This task expects students to use the assembled, animating hierarchy to determine the bounding boxes and draw them in their own coordinate system so that the animating transformations move them correctly. Figure 1(c-d) show examples of bounding boxes highlighted in student programs.

Task 4-1: Moving lights: Implement the ability to add lights to any node in the graph. Use this functionality to add at least one light fixed to the amusement ride and another light that is stationary with respect to the scene.

This task simply underscores that lights can be treated as vertices and transformed similarly. Figure 1(e) shows a student example.

Task 4-2: Object-stationary camera: Implement the ability to attach a camera to any node in the scene graph. Introduce a globally stationary camera and implement the ability to switch between the two cameras at the press of a button.

This task expects students to work out the world-to-view transformations by first principles. Often students question why they must know the math behind provided camera-setting functions like `gluLookAt` (from the GLU library) or `glm::lookAt` (from the glm library [glm14]). Implementing a moving camera presents a classic challenge to realize the limitations of provided functions, what transformations they provide and how to assemble them from first principles. Figure 1(f) shows a screen shot captured from the moving camera for the model in Figure 1(e).

Task 4-3: Keyboard-based camera control (extra credit): Implement keyboard controls for zooming, panning and rotating both cameras implemented in Task 4-2. No additional hints on how to implement this are provided.

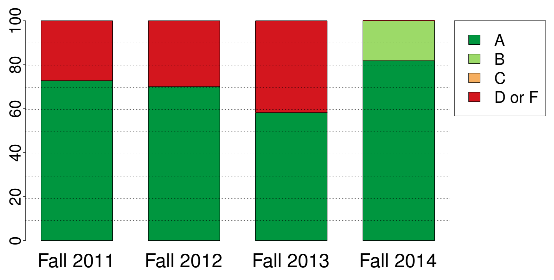
This task further expects students to identify that zoom, pan and rotation of cameras are the easiest to visualize in their own coordinate system. This makes these features independent of how the camera itself is moving.

3.3.1. Observations

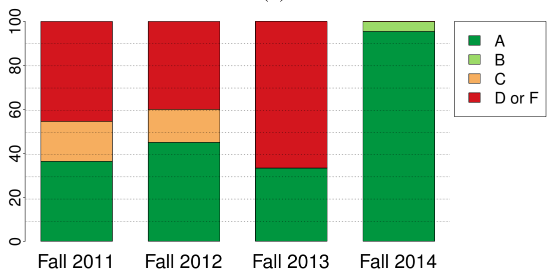
Figure 2(a,b) shows how students performed in two specific tasks across four semesters: creating a hierarchical model and animating it (Task 3-2). We hypothesized that students took time or had difficulty conceptualizing how a scene graph hierarchy can be used to model a complex model progressively and animating it in as desired. In Fall 2014 we introduced Task 3-1 that helped them with this understanding, before they created and animated the model. This showed a significantly better performance in these two specific tasks. Figure 2(c) shows the student performance in the new tasks that were assigned to them in Fall 2014 (the last three tasks were performed in pairs and both students got the same grade for these tasks).

3.4. Group work

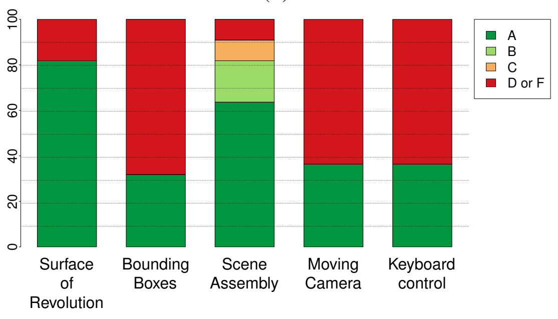
Our course did not allow any group work before Fall 2014 for any assignments. However students were allowed to discuss problems at a high level with each other. We often found students problem-solving collectively, borrowing ideas from each other and brainstorming solutions (and acknowledging them). In Fall 2014 we allowed students to work in pairs for the last 3 assignments, in the hope that they would reap



(a)



(b)



(c)

Figure 2: *Performance on the specific tasks of (a) creating a hierarchical model and (b) animating it in a specific way, completed individually in all semesters. We suspected that before Fall 2014, several students took time in creating the hierarchical model which led to completing the animation part partly or incorrectly. In Fall 2014 when we introduced scene graph modeling and manipulation more progressively, we believe students had an easier time completing both tasks and performed significantly better. (c) Student performance in the new tasks in Fall 2014 (the last three tasks were performed in pairs).*

the benefits of pair programming and group work (brainstorming math solutions themselves, programming and debugging, etc.). Thus, tasks 4-1 to 4-3 above were completed in pairs, after combining their individual amusement ride models into a single scene. Students continued to work in pairs for the ray tracing assignments. We correspondingly increased the expected outcomes of the group assignments so that students may achieve more (e.g. Tasks 3-3, 4-2 and 4-3 were never used together in a single semester). In order to ensure that each students was developing necessary skills and understand, we mandated that the first 3 assignments and the two exams be completed individually.

We observed three significant differences in the last half of the semester that we attribute to group work. We found that number of students seeking our help using office hours dropped significantly (it picked up again during the ray tracer). This was coupled by a retention or a modest increase in student performance for these assignments as compared to previous semesters, hinting that groups were working successfully. Secondly we informally observed a lot of face-to-face discussion between groups in the lab. Finally we believe this to be a contributing factor to the significantly better overall grade performance in the course (Figure 4), despite assigning more complex tasks within the same time frame.

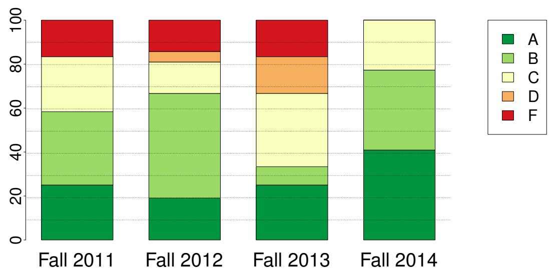


Figure 4: *Student grades across four semesters.*

4. Results and Discussion

We measure the effectiveness of our new approach using three primary questions that we believe fulfill the necessary outcomes of a challenging undergraduate CG course: (1) Do students know how to navigate a task "end-to-end" by working out Math and implementing the visual effect in a program? (2) Have students developed the ability to set up models and assemble transformations to produce a desired visual goal? (3) Were students able to achieve more in the semester and did they perform better while achieving them?

To verify students' ability to apply math directly or use transformations, we identified (possibly overlapping) parts of several assignments and the exams that were related to (1) applying math and (2) using transformations. Figure 3 shows the performance of students in these tasks, compared before and after our approach. This shows that students performed better in tasks that expected them to formulate transformations to complete visual tasks. However student performance did not improve in applying math directly to implement a visual effect as the semester progressed and compared to previous semesters, although it did not worsen.

Finally Figure 4 shows a significant percentage increase in the number of A and B grades in class as compared to previous semesters. We attribute this primarily to two factors: (i) a slower progression into various tasks leading to better performance in individual assignments, and (ii) group work in the later assignments. In summary, students achieved more, performed slightly better in piece-wise graded course work and significantly better overall in the course when we tried our new ideas.

5. Conclusions and future work

Although the above analysis is admittedly not the most rigorous, it provides preliminary evidence that students who otherwise struggle with math still perform well in a graphics course when given many chances to accomplish graphical tasks that require working with math. Students also perform better when such tasks are worked out in pairs, possibly boosting each other's understanding.

In the future we wish to expand this approach by assigning more work in the group assignments. We wish to try an "interactivity" approach to our assignments, where tasks related to interactive modeling would be assigned. An alternative approach may be to view such a graphics course as even more "applied" in nature, by making the material explicitly geared towards the usage of math in graphics [Len02]. A more long-term ambition is to couple our CG course more tightly with the supporting Math courses, possibly providing student material in the Math class related to how those concepts are practically applied in computer graphics.

6. Acknowledgments

The following students are acknowledged for writing the programs and creating the models that are shown in Figure 1:

Figure 1(a)	Nathan Gregg
Figure 1(b)	Nicholas Christensen
Figure 1(c)	Tyler Hasz
Figure 1(d)	Seth Davis
Figure 1(e-f)	Nicholas Christensen and Nathan Gregg

References

[ACM14] ACM CS curriculum, 2014. http://www.acm.org/education/CS2013-final-report.pdf. 2

[ACSS06] Angel E., Cunningham S., Shirley P., Sung K.: Teaching computer graphics without raster-level algorithms. In *Proc. SIGCSE* (2006), pp. 266–267. 2

[AK07] Amresh A., Karnick P.: Creating interest in computer graphics by teaching game development. *EG Education Papers* (2007), 9–16. 2

[Ang11] Angel E.: *Interactive Computer Graphics: A Top-down Approach with Shader-Based OpenGL*, sixth ed. Addison-Wesley, 2011. 4

[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 *Proc. Eurographics (Education Papers)* (2009), pp. 7–14. 2

[AS11] Angel E., Shreiner D.: Teaching a shader-based introduction to computer graphics. *IEEE Computer Graphics and Applications 31*, 2 (2011), 9–13. 4

[glm14] The opengl mathematics (glm) library, 2014. http://glm.g-truc.net/0.9.6/index.html. 4, 5

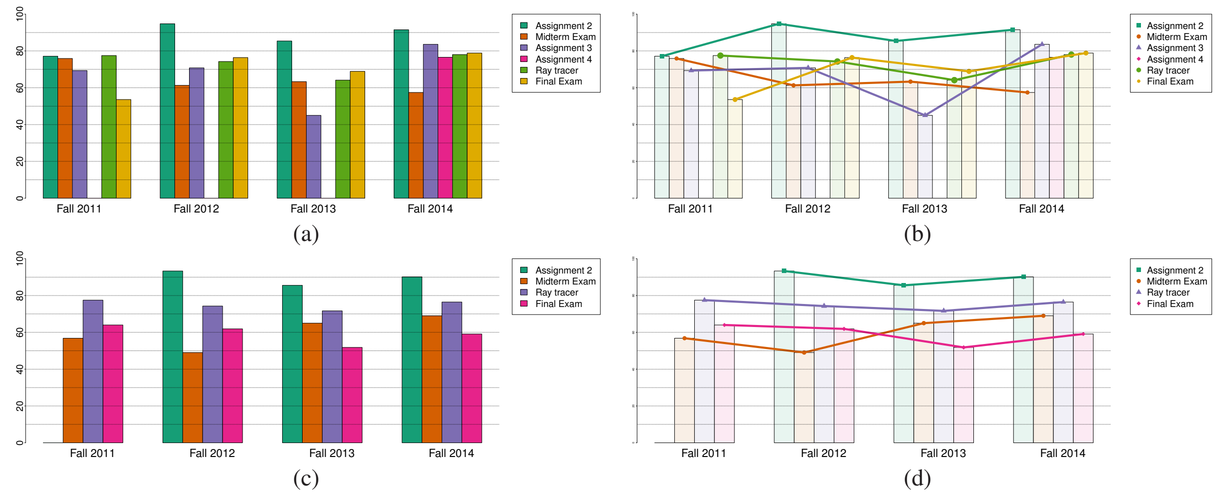


Figure 3: *Performance using average percentage scores on tasks related to (a-b) using transformations and (c-d) application of direct math. We select possibly overlapping parts of each graded item that pertained to these two categories. Some items in some semesters did not have components related to a category. (c) and (d) show the same results as (a) and (b), but highlight the trend of average performance in the same graded items across semesters. (a-b) the percentage scores in tasks related to using transformations, arranged chronologically in each semester. This illustrates whether students got better at formulating transformations as the semester progressed. (c-d) the percentage scores in tasks related to working out and apply math, arranged chronologically in each semester. This illustrates whether students got better at working with math to complete visual tasks as the semester progressed.*

[HS05] Hoetzlein R. C., Schwartz D. I.: Gamex: A platform for incremental instruction in computer graphics and game design. In *ACM SIGGRAPH 2005 Educators Program* (2005), SIGGRAPH '05. 2

[Len02] Lengyel E.: *Mathematics for 3D Game Programming and Computer Graphics*. Charles River Media, Inc., Rockland, MA, USA, 2002. 6

[ope11] Openscenegraph, 2011. http://www.openscenegraph.org/. 4

[Owe92] Owen G. S.: Teaching computer graphics using renderman. In *Proc. SIGCSE* (1992), pp. 304–308. 2

[PA14] Peters C. E., Anderson E. F.: The Four I's Recipe for Cooking Up Computer Graphics Exercises and Assessments. In *Eurographics 2014 - Education Papers* (2014), Bourdin J.-J., Jorge J., Anderson E., (Eds.). 4

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

[She13] Shesh A.: Toward a singleton undergraduate computer graphics course in small and medium-sized colleges. *Transactions on Computing Education 13*, 4 (2013), 17:1–17:21. 3

[SS03] Sung K., Shirley P.: A top-down approach to teaching introductory computer graphics. In *Proc. SIGGRAPH 2003 Educators Program* (2003), pp. 1–4. 2

[TBN06] Tori R., Bernardes Jr. J. a. L., Nakamura R.: Teaching introductory computer graphics using java 3d, games and customized software: a brazilian experience. In *Proc. SIGGRAPH 2006 Educators program* (2006). 2

[ymc14] Ymca dance, 2014. http://en.wikipedia.org/wiki/Y.M.C.A._(song). 3, 4