91.427/91.546 Computer Graphics I Spring 2015

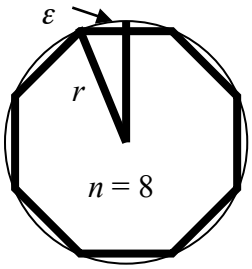
Final Take-Home Exam

Instructions (also distributed via piazza):

1. The exam will be available for download from Friday, May 1, 2015 08:00 AM until Friday, May 8, 2015 10:00 AM.
2. Download the exam file at http//:www.cs.uml.edu/~haim/teaching/cg/427-546/2015-spring/final/91-427-546_s2015_final_exam.pdf. When prompted, enter your CS userID and the password you submitted.
3. Record the date and time you downloaded the exam. If you download multiple times, only the FIRST time counts, so that's the one you will need for your submission (see #4 below).
4. The exam is made of a collection of short programs. Each one should be submitted in its own separate file, named as instructed in the exam document. Ideally, you should wrap them all into a Web page, from which we can see each program's code (and try it, if it works).
5. Put all your submission files in an archive.
6. Name it *yourLastName_yourFirstName_91427546s2015_final_YYYY-MM-DD-HH-MM.zip* (or *.gzip*, or *.gz*) , referring to **your** last name, underscore, **your** first name and the date and time (in 24 hr format) you downloaded the exam; (mine would be *levkowitz_haim_final_91427546s2015_final_2015-04-30-20-07.zip* if I downloaded at that day/time.).
7. Note that you are **NOT required to actually implement these programs**, i.e., I do not expect you to test them; however, an actual implementation might help you make sure you got it right. If you do get a program to run, please let us know and we'll try it out too; it might help with your grade if your program isn't that clear, but it does what it was supposed to do. Please append –tested to the filename of any program you actually tested.
8. The exam is **due** no later than **72 hours from the moment you downloaded it. If your exam is late, it will NOT be graded and it will get a grade of 0 (ZERO). No exceptions!**
9. Submit will accept submission until **Friday, May 8, 2015, 10:00 AM.**
10. When ready to submit, upload your archive to CS and cd to the directory where it is located.
11. Use the **`submit`** command as follows:
    `submit haim 91427546s2015-final` *yourArchiveFileName*
12. The exam has a total of 170 points. I will add 10 extra-credit points to each question that does not specifically require an interactive program, if you make your program interactive.
13. **REMEMBER: You must submit the final no later than 72 hours from the moment you downloaded it.**
14. **REMEMBER (also): you are to do this exam entirely on your own.** You are

allowed to consult your own notes, and any book. Any hint of collaboration, and / or the use of any other sources (such as solutions on Web sites, other people, etc.) will immediately result in any or all of the following: 0 on this exam; F in the course; potential disciplinary actions (see my Academic Honesty Guidelines page at http://www.cs.uml.edu/~haim/teaching/admin/academic_honesty.html for details). If you doubt my ability to detect violations of these rules, I urge you, beg you, for your own good, to rethink this doubt.

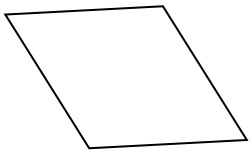
1. (**20 points**) Write a program that will take two line segments in parametric form as input, and will return their intersection point, if they intersect. If they don't, the program should return an appropriate response. (One parametric form of a line equation is: $p(t) = (1 - t)\, p_1 + t\, p_2$. To input the two lines, you will have to input (x, y, z) coordinates for each one of the two endpoints, p_1 and p_2 for each one of the lines. If you prefer, you can input them interactively on the screen. Submit in a separate file `1-line-line-intersection.js` (or `1-line-line-intersection-tested.js`).
2. (**10 points**) Extend the program in the previous question to find the intersection of a line segment with a planar polygon. Submit in a separate file `2-line-polygon-intersection.js` (or `2-line-polygon-intersection-tested.js`).
3. (**20 points**) When a circle is represented by its inscribed polygon, a tolerance ε is defined as the maximum perpendicular distance from the arc to the chord between two points on the circumference of the circle. Derive the relationship between the number of points n, the radius r of the circle, and the tolerance ε. (See example in figure.) Write a program that will take as inputs the number n and the radius r and will draw an n-sided polygon approximation to a circle of that radius, and the circle. The program should report the tolerance ε. Get **extra 10 points** for implementing an interactive slider to input n and get direct manipulation of the polygon. Submit in a separate file `3-polygon-circle-approx.js` (or `3-polygon-circle-approx-tested.js`).



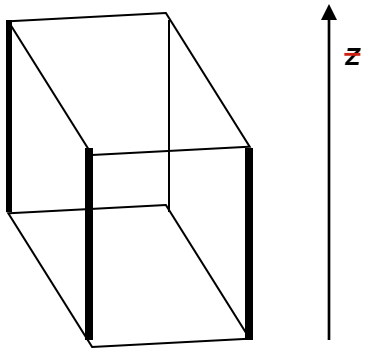
4. (**30 points**) Write a program to display a rotating cube inside a box with three light sources. Each light source should cause the projection of the cube onto one of the three visible faces of the box. Submit in a separate file `4-projected-cube.js` (or `4-projected-cube-tested.js`).

5. (**20 points**) Write a program to calculate the plane equation coefficients, given *n* vertices of a polygon. The vertices are enumerated in a counterclockwise direction. The polygon may be planar or just approximately planar. Your program's output should be the plane equation (in the form of [A B C D], the plane's normal) and a flag indicating whether the plane is **exactly** or only **approximately** planar. Submit in a separate file `5-plane-from-polygons.js` (or `5-plane-from-polygons-tested.js`).

6. (**20 points**) Write a program to generate a right prism from any planar polygon. Thus, starting from

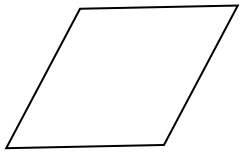


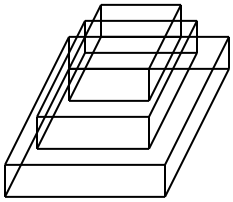
we get, for example



Your input should be a computer model of the polygon (e.g., vertices, or edges, per your choice) and a *z* height, and your program should output a computer model of the prism. (This is modeling, not drawing, but you may, optionally draw your generated model, for additional 5 points.) Submit in a separate file `6-prism.js` (or `6-prism-tested.js`).

7. (**20 points**) Write a program to create the following tower of 3-D object: start with any planar 2-D polygon in the X-Y plane (e.g., the one above). Create a prism (using the above program). Other inputs should be HOWMANY (how many layers), THICKNESS (how thick is each layer, this is the *z* height in the previous program), and SHRINK (the percentage shrink factor in each layer). Submit in a separate file `7-tower.js`. (or `7-tower-tested.js`).





8. (**30 points**: 6 points each item below) write an interactive program that will let the user
   a. sample the color of a pixel pointed to;
   b. select a brush shape (e.g., circle or square);
   c. assign the sampled color to the brush;
   d. free-hand draw with the brush;
   e. erase with the brush.

   Submit in a separate file `8-colors.js`. (or `8-colors-tested.js`).