Comp.4270/Comp.5460 (91.427/91.546) — Computer Graphics I — Spring 2017

Final Exam (Take-home)

Dr. Haim Levkowitz

Wednesday, April 26, 2017

Last Name:	
First Name:	
yourEmail at student.uml.edu:	
yourEmail at cs.uml.edu	

Instructions — read carefully!

1. **Due back:** no later than **72 hours after** first download or **Tuesday, May 2, 2017, 11:59 pm** (whichever is earlier).

2. The exam has a total of 150 points.

3. Each program should be submitted in its own separate file, named as instructed in the exam document. Ideally, you should wrap them all into a "landing" Web page, from which we can see each program's code (and try it, if it works).

4. Put all your submission files in an archive.

5. As in all your previous submissions, you **MUST** include a disclosure of **all resources** you have used, where you got them, how much you used, if/how much you modified. Include it as a comment at the **TOP of each source file**. Absent such detailed comment at the beginning of the file, I will not look at the rest of it and your grade for this file will be zero (0).

6. Name it `Last-First_427546s2017_final_YYYY-MM-DD-HH-MM.zip` (or `.gzip`, or `.gz` — **please NO** `.rar`), where Last = your last name; First = your first name, and the date and time (in 24 hr format) you downloaded the exam; (mine would be `levkowitz-haim_427546s2017_final_2017-04-30-20-07.zip` if I downloaded on April 30, 2017 at 8:07 pm).

7. Note that you are NOT REQUIRED to actually implement these programs. BUT, an actual working implementation is worth an extra 10%! above and beyond the number of points a question is worth. 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 is not 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 the FIRST time. If your exam is late, it will NOT be graded and it will get a grade of 0 (ZERO). No exceptions!** (unless you have compelling, documented reasons and have made prior arrangements with me).

9. Submit will accept submissions until **Tuesday, May 2, 2017, 11:59 pm**, but I do encourage you to not wait till the last moment: submit can fail you if too many people are trying to submit at the same time.

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 427546s2017-final yourArchiveFileName`

12. **REMEMBER: You must submit the final no later than 72 hours from the moment you first downloaded it.**

13. **REMEMBER (also): you are to do this exam entirely on your own.** You are allowed to consult your own notes, any book, any other resource, but you must disclose any and all such resources as described above. 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).

Visible surface determination — 50 points

1. (20 points) Write a program that will take as input a prism by its four vertices (v_1, v_2, v_3, v_4) and a cylinder by its center, radius, and height (c, r, h), and will draw them with their right relative visibility. Input: (v_1, v_2, v_3, v_4), (c, r, h). Output: Draw the three objects with the correct visibility (see Figure 1).

2. (20 points) Add a cone (by its center, radius, and height (c, r, h), see Figure 2) .

3. (10 points) What about adding an arbitrary object?

Submit in a separate file `1-visible-surface.js`
(or `1-visible-surface-tested.js`).

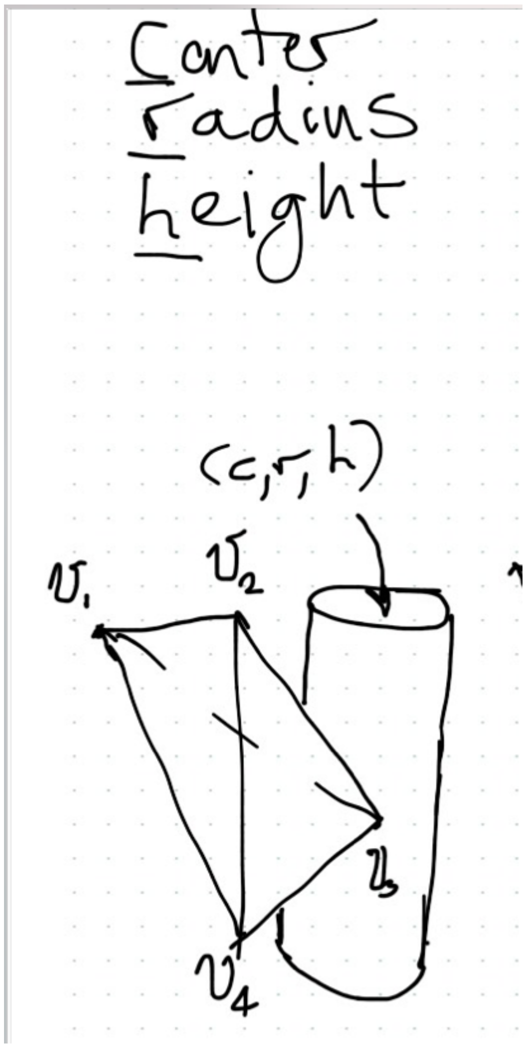

Figure 1: A prism and a cylinder

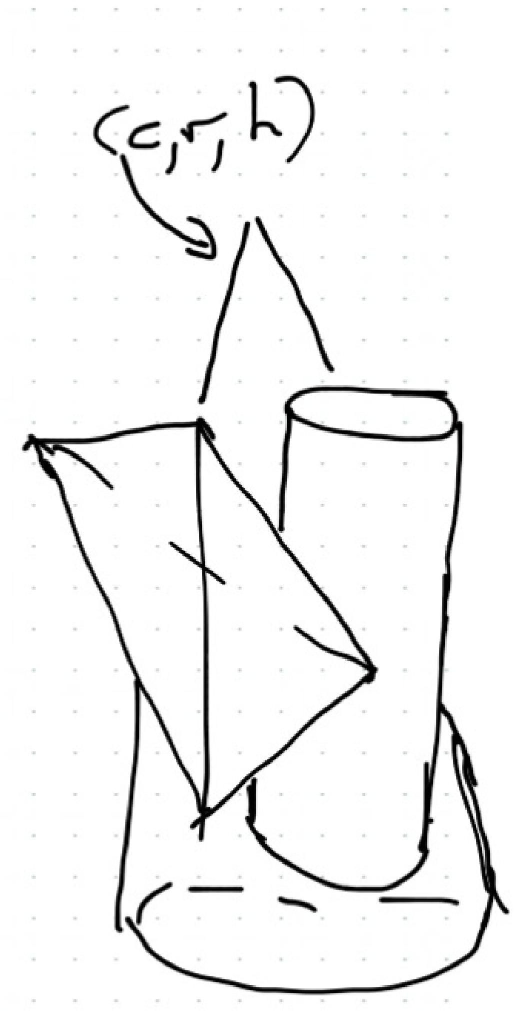

Figure 2: A prism, a cylinder, and a cone

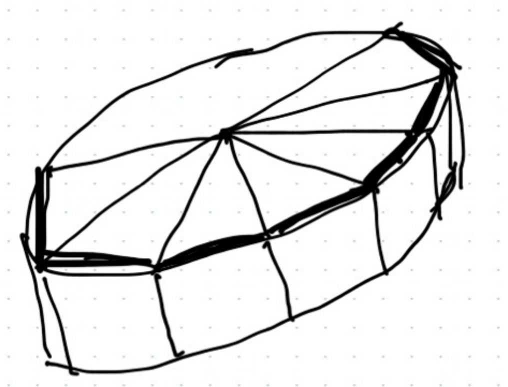

Figure 3: Cake

Approximating objects with polygons — 30 points

A world-famous bakery has hired you to render their new line of cakes. The cakes are round. Your renderings need to depict the entire cake as well as slices within it (see Figure 3). You decide that it would be much easier to render a cake as a polygonal object rather than a round one. When you tell your client, they agree, but on one condition: you must provide a difference between the weight of a complete round cake, and the weight of your polygonal approximation as a function of the number of slices that are being rendered. To assist you with your calculations, they will give you the cake's radius r, its height h, and the relationship w_v between the volume v and the weight w of the cake.

1. (5 points) Derive the relationship between the number of slices n, the radius r of the cake, and the weight w . (Hint: you will need to figure out ϵ, the difference between the true round object and its polygonal approximation — see Figure 4.)

2. Write a program that will

   (a) (5 points) take as input the cake's radius r, height h, the relationship w_v, and the number of slices n, and report the weight of the approximation as a percentage of the true weight of the cake;

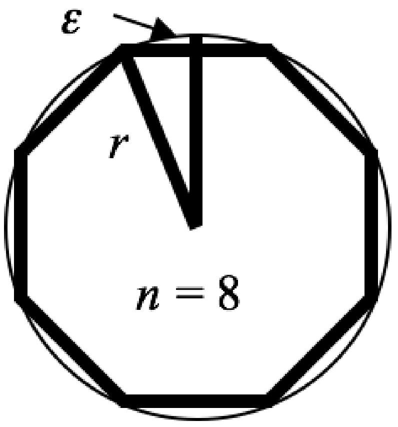

Figure 4: Cake geometry

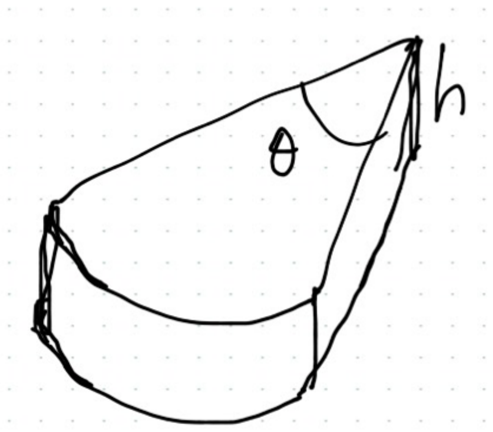

Figure 5: Cake slice, original

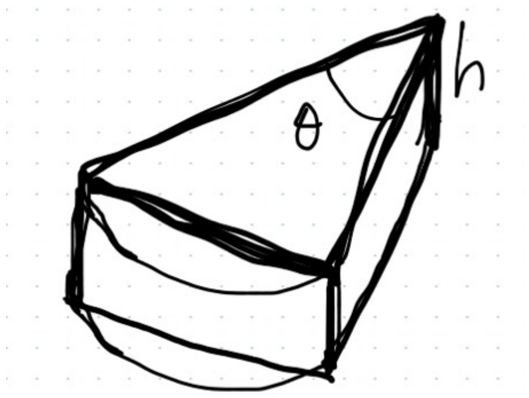

Figure 6: Cake slice, approximated

(b) (5 points) take as input the difference ϵ as a percentage of the radius r, and report the number of slices n that will provide the closest approximation;

(c) (5 points) take as inputs the center c (in 3D), the radius r, an angle $0 \leq \Theta \leq 360°$, a height h, and an ϵ difference, and draw a Θ slice (Figure 5) and its polygon approximation (Figure 6).

(d) (10 points) "implement" interactive inputs and get direct manipulation of the drawing (i.e., get the drawing to be updated "live" as input values change).

Submit in a separate file `3-cakes.js` (or `3-cakes-tested.js`).

Clipping — 30 points

Write a program to clip an object against a given view volume. You are free to choose whichever object and clipping algorithms you prefer.

Input: 1. View volume; 2. object.

Output: a drawing of the view volume (in light/dashed lines) and the clipped object (or whatever of it that survived) in thicker, solid lines.

1. (10 points) Perspective view volume.
2. (10 points) Parallel view volume.
3. (10 points) An interactive input is worth these extra 10 points.

Submit in a separate file `4-clipping.js` (or `4-clipping.js`).

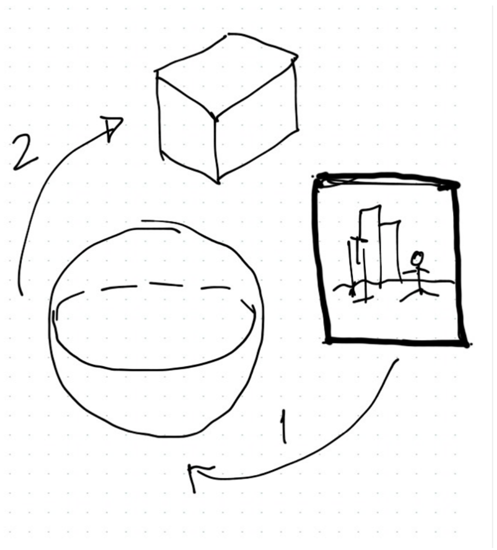

Figure 7: Texture mapping

Texture and shading — 40 points

Write a program that:

1. (10 points) Reads as input: a sphere, a cube, and an image; these can be read from files (see Figure 7).
2. (10 points) Maps the image to the sphere.
3. (10 points) Environment-maps the sphere to the cube (see Figure 8).
4. (10 points) "Bumps" the cube (see Figure 9).

Submit in a separate file `4-texture-shading.js` (or `4-texture-shading-tested.js`).

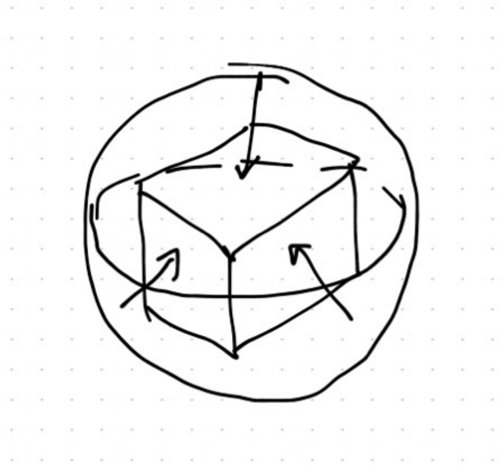

Figure 8: Environment mapping

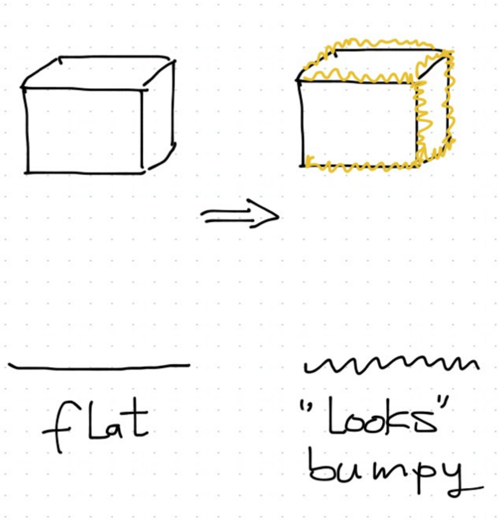

Figure 9: Bump mapping