Final Exam (Fall 2021)

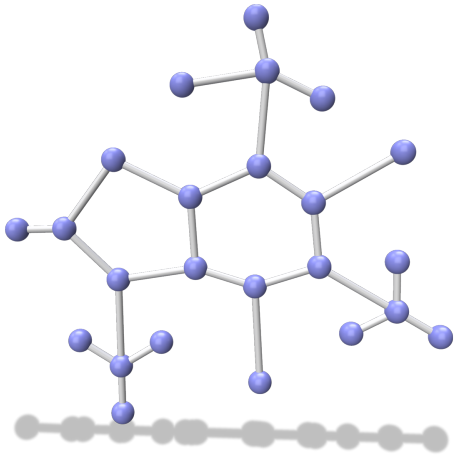
CMU 15-462/662

Instructions:

- Answers must be submitted by **Monday, December 13 2021 at 11:59:59 AM Eastern time** (note that 11:59am is right before lunch—not midnight!).
- There is no time limit; you may work on the exam as much as you like up until the deadline.
- This exam is **open book, open notes, open internet**, but you *must work alone*[1].
- You are free to type solutions or write them by hand. Rather than write on the exam directly, you should write on separate pages of paper, with question numbers clearly indicated. Typed solutions should use mathematical typesetting for formulas and equations (e.g., via LaTeX or the Equation Editor in MS Word/Google Docs); equations written in plain text will not be accepted.
- **You must write in complete sentences to receive any credit!** A "complete sentence" means, at bare minimum, proper capitalization, punctuation, spelling, and grammar. Sentence fragments, sentences written in all-lowercase letters, etc., will receive *zero points*. If you need any further guidance on how to write answers as complete sentences, see the exercise solutions (in gray boxes) provided throughout the semester.
- Your solutions *must* be submitted by uploading a PDF via Gradescope. We will *not* accept any other form of submission (such as physical printouts, email, Piazza, etc.). For the coding portion, you must submit a single file `MolecularDynamics.cpp` via Autolab.

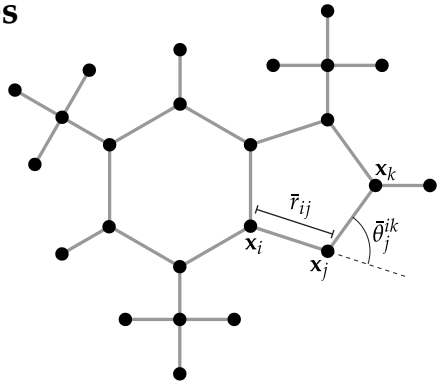
Problem	Your Score	Possible Points
1.1		7
1.2		6
2.1		5
2.2		10
2.3		5
2.4		25
3.1		5
3.2		5
3.3		5
3.4		6
3.5		5
3.6		5
3.7		5
3.8		6
Total		100

[1] "Alone" means you cannot discuss the exam with your classmates, your friends, your dog, your cat, or any other creature containing deoxyribonucleic acid.



Throughout the semester we've been hinting at the fact that basic knowledge from computer graphics can be applied to a much broader set of problems in computer science, scientific computing, machine learning, engineering, computer vision, quantitative finance, etc. In this exam you'll work through one topic—molecular dynamics—that has the exact same "look and feel" as a computer graphics problem, but is broadly applicable to real-world challenges like chemical engineering, drug discovery, or understanding diseases like COVID-19. By putting a graphics spin on this question, you'll also develop some tools that are useful for visualizing and communicating results from this domain. Along the way we'll encounter topics from all the major areas of the course: geometric modeling, animation, and rendering/rasterization. Let's get started!

1 Modeling Molecules



When building up any kind of graphics program, one of the first questions we often have to ask is: what data structure(s) should be used to represent our data? This choice will have a big impact on the efficiency and flexibility of algorithms down the line.

In our case, we want to represent *molecules*, and for our purposes we'll adopt a simple "ball and stick" model, i.e., a molecule will be viewed as a collection of *atoms*, connected by *bonds*. The most basic thing we need to keep track of for each atom i is its *position* $\mathbf{x}_i \in \mathbb{R}^3$. For the purpose of simulation we will also need to store other data like the *velocity* $\mathbf{v}_i \in \mathbb{R}^3$, the total *force* $\mathbf{f}_i \in \mathbb{R}^3$, and some constants like its *mass* $m_i \in \mathbb{R}_{>0}$ and *charge* $c_i \in \mathbb{R}$. Each bond is given by a pair of atoms i, j, as well as an equilibrium distance $\bar{r}_{ij} \in \mathbb{R}_{>0}$. Finally[2], molecules also have equilibrium *bond angles* $\bar{\theta}_j^{ik} \in [0, \pi]$, which describe the exterior angle between segments ji and jk. These quantities are annotated in the figure above.

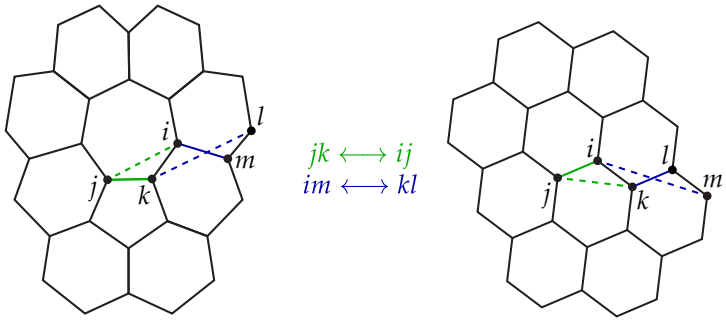
[2] Actually, molecular models also often consider equilibrium *torsions* associated with tuples of four atoms—for simplicity,we'll omit this aspect of our model.

QUESTION 1.1 — A MOLECULAR DATA STRUCTURE [7 POINTS]

Describe a data structure that could be used to encode all of this information, in a way that's easy to access during simulation. Be as specific as possible, and make sure to explain how you would store *all* the data. You should assume in particular that bonds might break apart or get connected during simulation. There is not just one right answer! However, the asymptotic cost of accessing information or modifying your data structure should not be unnecessarily large—for instance, it must not take $O(n)$ time in the number of atoms n to determine the neighbors j of a given atom i.

[Hint: This question should remind you of our discussion of the design of mesh data structures. However, you cannot just write "I would use an adjacency list/incidence matrix/halfedge mesh" since those are data structures for polygon meshes—not molecules! If you choose to go down this path, you need to explicitly describe how those data structures can be adapted to encoding molecules, and what the differences are relative to storing a polygon mesh.]

QUESTION 1.2 — DISLOCATING DATA [6 POINTS]



Graphene is a large molecule made up entirely of carbon atoms, which like to arrange themselves into a perfect hexagonal grid. One way a graphene molecule can move into a lower-energy state is to replace a neighboring pentagon-heptagon pair with a more regular hexagon-hexagon pair. As depicted above, this transformation can be achieved by replacing a bond between atoms j, k with a bond between i, j, and a bond between atoms i, m with a bond between k, l. Sketch out in words how you would implement this transformation, using your data structure from the previous question. You do not need to write any code, but you do need to give a description at a level that is easily implementable. (For instance, you *cannot* simply say, *"I would first swap bond jk with bond ij, then swap bond im with kl"*!)

2 Animating Atoms

Now that we have a data structure, let's simulate the motion of our atoms over time. Basically the molecule should wiggle and jiggle due to kinetic energy (i.e., heat!), while also trying to retain a basic shape (given by bond lengths and angles).

In any molecule, there are several different forces acting on the atoms—each of which arises from the gradient of a different potential energy:

- **Stretching Potential.** Each bond ij between atoms i and j seeks to achieve an equilibrium rest length $\bar{r}_{ij} \in \mathbb{R}_{>0}$. Specifically, the molecule seeks to minimize the energy

$$\mathcal{E}_{\text{stretch}} := \frac{1}{2} k_{\text{stretch}} \sum_{ij \in \text{bonds}} (r_{ij} - \bar{r}_{ij})^2,$$

  where bonds is the set of bonds, $k_{\text{stretch}} \in \mathbb{R}_{>0}$ is a constant which determines the strength of the stretching term, and $r_{ij} := |\mathbf{x}_i - \mathbf{x}_j|$ is the current distance between atoms i and j.

- **Bending Potential.** Triples of atoms i, j, k also seek to position themselves so that the exterior angle between bonds ji and jk is equal to a given equilibrium rest angle $\bar{\theta}_j^{ki}$. Specifically, the molecule seeks to minimize

$$\mathcal{E}_{\text{bend}} := \frac{1}{2} k_{\text{bend}} \sum_{ijk \in \text{corners}} (\theta_j^{ik} - \bar{\theta}_j^{ik})^2,$$

  where corners is the set of corners between two bonds incident on the same atom, k_{bend} is a constant which determines the strength of bending, and θ_j^{ik} is the current angle between bonds ji and jk (which in turn depends on the current positions $\mathbf{x}_i$, $\mathbf{x}_j$, $\mathbf{x}_k$).

- **Nonlocal Potentials.** What really makes molecular dynamics interesting is a *nonlocal* potential: each atom exhibits a force on every other atom, whether or not they are connected by a bond. In particular, there are two terms. The first is a *Coulomb potential*, which accounts for electrostatic repulsion, i.e., atoms pushing each other away due to their net charge:

$$\mathcal{E}_{\text{Coulomb}} := k_{\text{Coulomb}} \sum_{\substack{i,j \in \text{atoms} \\ i \neq j}} c_i c_j \frac{1}{r_{ij}}.$$

  Here atoms is the set of atoms, $k_{\text{Coulomb}} \in \mathbb{R}_{>0}$ is a constant controlling the strength of the Coulomb term, c_i, c_j are the charges of atoms i and j, and as before, $r_{ij} := |\mathbf{x}_i - \mathbf{x}_j|$ is the current distance between atoms i and j. Note in particular that the sum is over all pairs of *distinct* atoms $i \neq j$, since otherwise one of the terms would be infinite (due to division by zero). Similarly, there is a so-called *Lennard-Jones* potential, which also accounts for attractive interactions:

$$\mathcal{E}_{\text{LJ}} := 4\epsilon \sum_{\substack{i,j \in \text{atoms} \\ i \neq j}} \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right).$$

  Like the Coulomb potential, the sum here is taken over all pairs of distinct atoms. The parameter $\varepsilon \in \mathbb{R}_{>0}$ controls the strength of the Lennard-Jones term, and the parameter $\sigma \in \mathbb{R}_{>0}$ controls the radius of influence.

The overall potential energy of the system is then

$$\mathcal{E}_{\text{molecule}} = \mathcal{E}_{\text{stretch}} + \mathcal{E}_{\text{bend}} + \mathcal{E}_{\text{Coulomb}} + \mathcal{E}_{\text{LJ}}.$$

Importantly, notice that all of these potentials depend only on the positions $\mathbf{x}_i$, and not the velocities $\mathbf{v}_i$.

QUESTION 2.1 — GO TO GREAT LENGTHS [5 POINTS]

A basic quantity in several of our potentials is the distance $\mathbf{r}_{ij} := |\mathbf{x}_i - \mathbf{x}_j|$ between two atoms. Give an expression for the gradient of $\mathbf{r}_{ij}$ with respect to $\mathbf{x}_i$, and also with respect to $\mathbf{x}_j$. Explain why your expression makes sense from a geometric point of view.

[Hint: Make sure to do a sanity check on your final expression, by thinking about what its type should be (a scalar, a vector, etc.)]

QUESTION 2.2 — MAY THE FORCE BE WITH YOU [10 POINTS]

In order to simulate our molecule, we will need to know the force corresponding to each potential. In this case, for any potential energy $\mathcal{E}$, the corresponding force on a given atom i is equal to *minus* the gradient with respect to $\mathbf{x}_i$:

$$\mathbf{f}_i = -\nabla_{\mathbf{x}_i} \mathcal{E}.$$

Equivalently, the three components of the force are given by the partial derivatives of $\mathcal{E}$ with respect to the three components of $\mathbf{x}_i$ (but it's typically much easier to compute the gradient all at once, rather than one coordinate at a time!).

a. Give an expression for the force on atom i due to the stretching potential $\mathcal{E}_{\text{stretch}}$.

b. Give an expression for the force on atom i due to the bending potential $\mathcal{E}_{\text{bend}}$. To make life easier for you, here's the gradient of an interior angle α made by the vectors $\mathbf{u} := \mathbf{b} - \mathbf{a}$ and $\mathbf{v} := \mathbf{c} - \mathbf{a}$ between points $\mathbf{a}, \mathbf{b}, \mathbf{c}$:

$$\begin{aligned} \nabla_{\mathbf{b}} \alpha &= -(\mathbf{w} \times \mathbf{u})/|\mathbf{u}|^2 \\ \nabla_{\mathbf{c}} \alpha &= (\mathbf{w} \times \mathbf{v})/|\mathbf{v}|^2 \\ \nabla_{\mathbf{a}} \alpha &= -(\nabla_{\mathbf{b}} \alpha + \nabla_{\mathbf{c}} \alpha) \end{aligned}$$

where $\mathbf{w} := (\mathbf{u} \times \mathbf{v})/|\mathbf{u} \times \mathbf{v}|$ is the unit normal of the plane containing the three atoms. You may use these terms in your final expression (you do not have to expand them).

c. Give an expression for the force on atom i due to the Coulomb potential $\mathcal{E}_{\text{Coulomb}}$.

d. Give an expression for the force on atom i due to the Lennard-Jones potential $\mathcal{E}_{\text{LJ}}$.

[Hint: It should be possible to evaluate your final expressions using nothing more than the atom positions $\mathbf{x}_i$, and any constants appearing in the original potential expressions. However, you do not *need to write one big, long, ugly expression purely in terms of $\mathbf{x}$! It is much better if you (clearly) define some intermediate quantities, and use those quantities in your final expression—just as you would when writing "nice" code. For instance, it is perfectly fine to use the quantity $\mathbf{r}_{ij}$ in one of your force expressions, so long as you define what this quantity means.]*

QUESTION 2.3 — DON'T LOSE ENERGY! [5 POINTS]

In our discussion of ODEs and time integration, we considered three different schemes: forward Euler, backward Euler, and symplectic Euler, and said that symplectic Euler was nice because you don't have to worry about it losing or gaining (much) energy: even over a very long time, the energy will just oscillate up and down a little bit.

Suppose for atom i you know the current position $\mathbf{x}_i \in \mathbb{R}^3$, the current velocity $\mathbf{v}_i \in \mathbb{R}^3$, and the force $\mathbf{f}_i \in \mathbb{R}^3$ due to all the potentials, as well as the atomic mass $m_i \in \mathbb{R}_{>0}$. Remembering that the "animation equation" is just *force equals mass times acceleration*, write down how you would compute the new position $\mathbf{x}_i'$ and velocity $\mathbf{v}_i'$ using symplectic Euler, assuming you want to take a time step of size $\tau \in \mathbb{R}_{>0}$.

[Hint: Remember that velocity is the time derivative of position, acceleration is the time derivative of velocity, and a time derivative can be written as a difference in two values, divided by a time step.]

QUESTION 2.4 — FORCE CODE [25 POINTS]

A lot of your time this semester was spent implementing graphics algorithms in C++. Let's put those skills to the test by seeing if... you can implement graphics algorithms in C++! We've provided a code skeleton for you in the files `MolecularDynamics.h` and `MolecularDynamics.cpp`, with several routines to complete. This code has no external dependencies, you should be able to compile it and run it if you wish to use the checks below.

> **IMPORTANT: You are not required to build and run the code unless you think it would be helpful!** You can just implement the routines, and hand in the code. As long as the logic of the code is basically correct, you will get full credit. It is not critical to get the code to the point where it is compiling and producing correct output. *We simply want to see that you know how to translate math into code.*

The code uses a data structure for a molecule that is likely (much) simpler than the one you described in Question 1.1, because it *assumes that the connectivity of the molecule does not change*. In particular, the `Molecule` class keeps track of

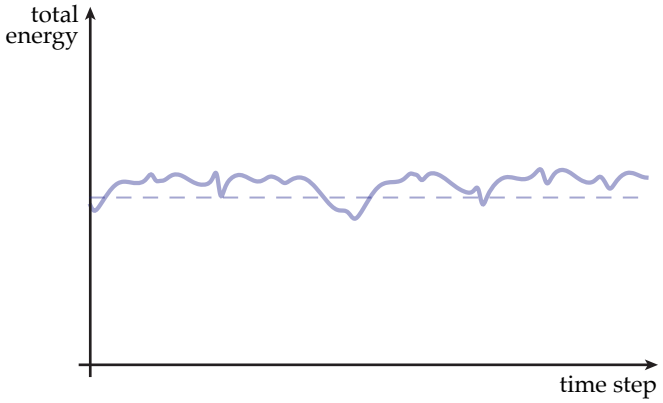
- a list `atoms` of the basic data for each atom (position $\mathbf{x}$, velocity $\mathbf{v}$, force $\mathbf{f}$, mass m, and charge c).
- a list `bonds` of the bonds between pairs of atoms i, j, and the associated equilibrium rest length $\bar{r}_{ij}$.
- a list `corners` of the atom triples i, j, k that seek to achieve an equilibrium rest angle $\bar{\theta}_j^{ik}$.

You must implement all the routines marked `TODO`, which encompass:

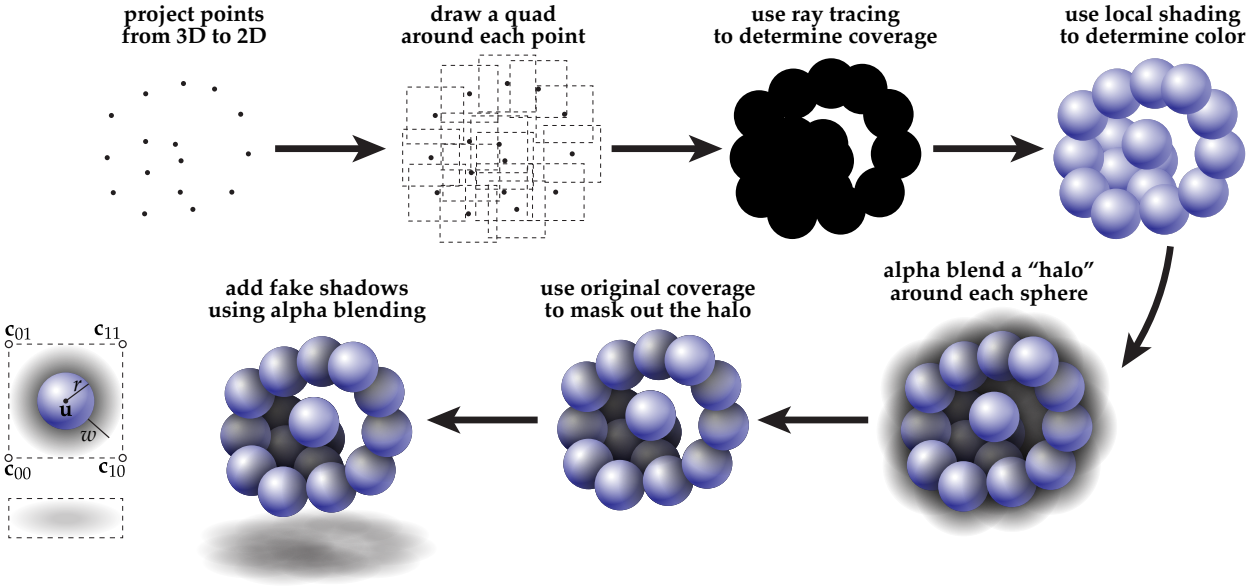
- evaluation of the potentials,
- evaluation of the forces, and
- time integration via symplectic Euler.

*[Note: it does <u>not</u> matter whether you got the force expressions right or wrong in Question 2.2. You will get full points as long as (i) you made an earnest attempt in Question 2.2 (e.g., you didn't just write "*all the forces are zero!*") and (ii) you correctly translated your force expressions to code. Basically we just want to check that you know how to convert mathematical expressions into code—which is an absolutely essential skill for almost any problem in computer graphics!]*

Sanity checks. If your code is working properly, then the total energy returned by `Molecule::totalEnergy()` should not drift by a significant percentage from the original value, even over very long integration times (as shown in the plot below). To help you verify the correctness of your derivatives, there is also a method `Molecule::checkDerivatives()` that will numerically validate your derivative methods against your potential methods. The worst error should be reasonably small (no bigger than about 0.001 or so, but don't get too hung up on this exact value).



3 Visualizing the Solution



After working hard on all that code, it's pretty unsatisfying to not be able to see what your evolving molecule looks like. If you think back to our very first lecture, that's what computer graphics is all about—making digital information visible! Let's think through how we could nicely display our molecules on screen, using a mix of ray tracing and rasterization techniques.

Our goal is to draw each atom as a sphere, and each bond as a cylinder. To keep things simple, we'll ignore perspective distortion, and just use an orthographic projection to get our 3D atom positions onto the 2D screen. For each projected atom, we'll draw a quadrilateral around the center of the projected position. We'll then use ray tracing to evaluate whether each pixel within the quad is covered by the sphere, and which point $\mathbf{p}$ in 3D space corresponds to the covered point. To determine the color of each pixel, we'll do some basic local lighting calculations using the location of the hit point $\mathbf{p}$. We'll then use a cheap "hack" to approximate ambient occlusion, making it easier to determine which atoms are in front, and which are in back. Finally, we'll use another cheap hack to simulate shadows below the atoms. The final effect is shown in the last stage of the "molecule rendering pipeline" depicted above.

QUESTION 3.1 — ATOMIC SUBROUTINE [5 POINTS]

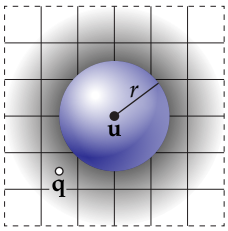
Suppose you're given a point $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$, corresponding to the center of an atom. Describe *any* orthogonal projection into 2D coordinates $\mathbf{u} = (u, v) \in \mathbb{R}^2$ (any projection at all!). Mathematically, what does it mean for a projection to be "orthogonal?" *[Hint: In general, what does it mean geometrically for a linear transformation to be orthogonal?]*

QUESTION 3.2 — QUERYING QUADS [5 POINTS]

Given a point $\mathbf{u}$ in the 2D plane, what are the locations of the four corners $\mathbf{c}_{00}, \mathbf{c}_{01}, \mathbf{c}_{10}, \mathbf{c}_{11}$ of the four corners of the *smallest* quad that will cover an orthogonally-projected sphere of radius r, plus a halo of width w?

QUESTION 3.3 — STABBING SPHERES [5 POINTS]

Suppose now that we want to test whether a pixel centered at a point $\mathbf{q} \in \mathbb{R}^2$ is covered by the sphere centered at $\mathbf{x} \in \mathbb{R}^3$, of radius r. (For now you can ignore the halo.) Beyond just determining coverage, we also want to know the location $\mathbf{p}$ of the nearest point of the sphere seen along the view ray through the pixel center. Give and describe the equation we have to solve in order to compute (i) whether the pixel is covered, and, if the pixel is covered, (ii) the location of the point $\mathbf{p}$. You should assume that we are working in the orthogonal projection from Question 3.1.



QUESTION 3.4 — SHADING SPHERES [6 POINTS]

Consider a pixel that *is* covered by a sphere, and let $\mathbf{p} \in \mathbb{R}^3$ be the closest hit point. As before, let $\mathbf{x} \in \mathbb{R}^3$ be the center of the sphere in 3D, and let r be the sphere radius. (Again ignore the halo.)

a. Write an expression for the intensity $I_{\text{diffuse}} \in \mathbb{R}_{>0}$ of a simple diffuse shading term, in terms of the position $\mathbf{p}$, the center $\mathbf{x}$, and the radius r. You may assume there is a single directional light source of intensity 1, with outgoing unit direction $\mathbf{l} \in \mathbb{R}^3$ relative to $\mathbf{p}$.

b. Write an expression for the intensity $I_{\text{highlight}} \in \mathbb{R}_{>0}$ of a simple Phong shading term, in terms of the same input data as in part (a). You may assume you are given a specular exponent $k \in \mathbb{R}_{>0}$.

c. Suppose $c_{\text{diffuse}} \in \mathbb{R}^3$ and $c_{\text{higlight}} \in \mathbb{R}^3$ are the RGB colors of the diffuse term and specular highlight. Write an expression for the overall RGB color $c \in \mathbb{R}^3$. (You do not need to incorporate any other terms, like ambient or fresnel.)

[Hint: You have all the information you need to figure out the view direction (up to translation of the image plane) and surface normal!]

QUESTION 3.5 — YOU HAD ME AT HALO [5 POINTS]

So far we basically have a shader for our quad that draws a solid sphere. Now let's add a semi-transparent halo around the sphere, that will darken everything behind it—this scheme is a cheap approximation of ambient occlusion, where objects are shadowed by the objects around them. Describe how you would set the color *and* alpha values of each pixel so that:

- pixels in the sphere are shaded as described in the previous question,
- the halo is black everywhere,
- the sphere appears completely solid, and
- the halo is semi-transparent near the sphere boundary, and completely transparent by the time it reaches the boundary of the quad.

The halo does not need to fall off at any particular rate, but should satisfy the criteria above.

QUESTION 3.6 — I'M OVER IT [5 POINTS]

Suppose we now draw our quads one at a time, using the "over" operator to perform alpha blending. Assume also that we do not use depth buffering at all, in any part of our pipeline. What do we need to do to ensure that the halo around atoms in the front properly occlude atoms in the back? Are there any situations where our cheap hack won't work very well?

QUESTION 3.7 — PUTTING ON A MASK [5 POINTS]

As seen in the bottom-right of the pipeline above, drawing a halo around every sphere darkens the atoms in the background—but it also puts an ugly black blur around everything. How can we use the original coverage information (from just the spheres, and not the halos) to "mask out" pixels that do not belong to any sphere? More precisely, suppose that for a single pixel $\mathbf{q} \in \mathbb{R}^2$ you are given

1. The color $c_0 \in \mathbb{R}^3$ of the current color of $\mathbf{q}$ stored in the framebuffer (i.e., in the image being rendered).
2. The color $c \in \mathbb{R}^3$ and alpha $\alpha \in [0, 1]$ of the pixel $\mathbf{q}$ in the quad currently being drawn.
3. The coverage $\beta \in [0, 1]$ at $\mathbf{q}$ determined from just the spheres (second image from the right in the top row of the pipeline figure).

Describe in words how you would use these values to compute the new pixel value. You do not need to give an explicit formula unless you *really* want to—for instance, it's fine to say things like *"then apply the over operator to this value to get that value"*.

QUESTION 3.8 — SQUISHY SHADOWS [6 POINTS]

Finally, we'll draw some fake shadows by drawing a second, squashed quad beneath each atom (see the lower, flatter dashed box in the pipeline figure above). This time, we won't hold your hand through every step of the process: describe how you would draw these shadows, giving as much detail as possible. You should specify things like: what kinds of transformations get applied, how do pixel colors and alpha values get computed, and where/how should these quads get drawn relative to the rest of the pipeline. Your answer should be between 1–2 paragraphs long. We're not looking for perfection here—rather, we're just looking for an indication that you know how to *think for yourself* when it comes to solving graphics problems!