CS 410, Fall 2018, Second Third of Semester Review
November 6, 2018

The following is not an exhaustive list of what has been covered, and to be very explicit, the Second Midterm may cover material discussed in class not been mentioned below. That said, hopefully the following will be a helpful as you review the broad topics and techniques we have covered since Midterm 1 in CS410.

1. Calculating where a ray and sphere intersect, and determining if they intersect, is an excellent time in computer graphics to introduce the general technique of mixing parametric and implicit forms in order to solve geometric problems. Consequently, you now are able to use the brute force approach to ray sphere intersection to illustrate in concrete terms this general approach.
2. There is a clever way of determining the intersection of a sphere and a ray. Since you have already implemented this as part of your programming assignments, you're aware that solving a general problem without the aid of a calculator would be annoying. That said, the construction of the algorithm is elegant and important, and you certainly could work out a concrete numerical example if the constituent components of the example were well enough chosen to make the ensuing arithmetic straightforward.
3. Earlier in the semester you were taught that there would be practical value in understanding what was described as the "at what point does a spaceship leaving earth come closest to Mars" problem. The clever algorithm for intersecting a ray in the sphere is exactly such an example. Be sure you understand and can explain the connection.
4. Three kinds of reflectance lie at the heart of computer graphics: ambient, diffuse (Lambertian), and specular. Your knowledge of all 3 is thorough and relates both to mathematical definitions and intuition with respect to identifying which of these are present in a given rendering of a scene. Your intuition is further bolstered by time spent experimenting with the SageMath notebooks which render illuminated scenes containing one or several spheres.
5. Ambient illumination may be thought of in several ways. First, is an extraordinarily convenient and simple hack. Second, a key element in realizing a believable rendering of many scenes. Third, a very coarse approximation to a common condition on the coast of California. (This last part might deserve a brief mention in the review section)
6. Two of the three kinds of reflectance require explicit light sources, and one does not. Be clear about which is which.
7. One and only one of the 3 types are reflections require knowledge about the relationship between the viewer (camera) and the position on the surface whose illumination is being computed. Be clear about which of the three types of illumination requires this extra information, i.e. direction to the camera.
8. You know how to compute the direction to a light source at [3, 4, 1, 0] specified in homogeneous coordinates. As you think about this light source specification recall programming assignment 3.
9. The computation of diffuse reflection from a point on a surface depends upon the intensity of the light source, the normal to the surface, and the vector indicating the direction from the point on the surface back to the light source. The exact equation and motivating diagram is something you can reproduce from memory.
10. Consider if you would agree with this statement: "The essence of Lambertian reflectance is the way photons are reflected back in the direction of the light source with greater intensity/frequency than they are reflected off to the sides." As you consider your response to the perhaps deceptively simple statement, it would probably be helpful to step through exactly what happens to a photon that hits a surface under the Lambertian model of reflectance.
11. For a light source located at a position in the world (x, y, z) and a point on a surface (sx, sy, sz), you can write down the precise values in a vector pointing back to the light source.
12. One might say of diffuse versus specular reflection that one is deep while the other is superficial. While this is admittedly a somewhat odd statement, it captures something of importance that you in turn can now explain.
13. Of the 3 forms of reflection, specular reflection involves the greatest amount of required information. In particular, it requires a vector representing the direction to the light source (also required for diffuse), a surface normal (again same as diffuse), a reflection Ray R that was not required for diffuse illumination, and finally a vector representing the direction to the camera/viewer. Of these, the reflection Ray R is arguably the most involved to compute. Since it is a fundamental element in computing specular reflection, you of course understand the construction of how it is expressed relative to the other known elements.

14. It seems the parameter alpha crops up in many places in computer graphics. It plays a critical role in Phong specular reflectance and is commonly assigned values such as 5 or 50, or even higher. You can explain precisely what alpha is doing both in words and with an equation.
15. There is an essential and very interesting aspect of specular reflection and its associated constant(s)k. Namely, there are 3D modeling packages/languages where k is expressed as a scalar rather than a three-tuple (three scalar constants). Given your understanding of specular reflection, you can now explain why only a single scalar value is a good choice in many circumstances. Likewise, you can explain the flexibility introduced by using a three-tuple.
16. Sometimes in 3-D modeling for computer graphics being "two-faced" is not such a bad thing. Be ready to explain what such an off-handed remark actually means.
17. Two concepts are easily confused. The first is the field of view of a pinhole camera. The second is the resolution, or pixel density, of an image. In a well-built rendering system, the controls used to modify the field of view are clearly separate from the controls used to change the pixel density. Having now built your own rendering system, you're familiar with this distinction, and therefore should have an easy time explaining it to others and fielding questions relating these two concepts.
18. Given the direction of a ray defined by a vector arriving at a point, as well as the 3D coordinates of the point, and finally the surface normal, you are able to choose between a series of alternative possible reflection rays provided the choices are qualitatively distinct. In other words, incorrect choices may be ruled out based on some clear conflict such as pointing in the overall wrong direction.
19. Both diffuse and specular reflection have dot products as key to their computation. In both cases, a negative value for the dot product has important geometric implications and also requires explicit handling in code. As you implement a ray tracer, and when you go to explain the process to others, you are comfortable now with the meaning and proper handling of those situations where the dot product yields a negative number.
20. For recursive ray tracing two distinct concepts have been introduced in general terms as well as precisely in SageMath code. These to concepts are attenuation and reflectivity. Having experimented with both in the SageMath notebook on recursive ray tracing discussed in lecture, you are comfortable with how each operates and how they are distinct from each other.
21. With all other camera and scene parameters left constant, you are now comfortable explaining what happens when recursion depth is increased. In particular, you can correctly argue the following proposition: with increased recursion the intensity of any given pixel must remain constant or increase.
22. You should be able to describe the high level and key distinctions between the projective pipeline versus ray tracing approaches to rendering 3D scenes.
23. Given its relative simplicity when compared to perspective projection, orthographic projection can be easily overlooked. However, you understand some of the important contexts in which it is most useful and you can define orthographic projection by writing down an example of an orthographic projection matrix.
24. A predator endowed with orthographic vision (orthographic projection) is not fooled by an elephant hiding behind a rabbit. Not so for standard issue predators whose eyesight follow the laws of perspective projection. You should have no trouble sketching an illustration of this admittedly somewhat off-the-wall approach to contrasting orthographic and perspective projection.
25. In the lecture introducing perspective projection, the first perspective projection matrix was derived for the case where the image plane is moved an amount d in front of the focal point. The second formulation placed the image plane at the origin with the focal point behind the image plane by an amount d. The corresponding perspective projection matrices look similar; swapping 1s and 0s between two positions yields one from the other. However, they behave differently and understanding these differences is important.
26. Some people in computer graphics choose to think of perspective projection as a linear operator, while others do not. It matters less who is right then that you can clearly state the crux of each side's argument. (It would be good to ask about this item in our review session since the issue has probably not received the attention it deserves.)
27. The terms "frustum" and "canonical view volume" are used nearly interchangeably when describing the perspective projection pipeline. Both terms now make sense to you, and you should be able to explain them in your own words and with your own simple illustrations.
28. A complete working model of the perspective projection pipeline has been presented using SageMath. Since you

have experimented with different camera specifications using this notebook, you can associate how an object – in our case a simple house – appears in the canonical view volume based upon alternative camera specifications.

29. In the perspective projection pipeline SageMath notebook you have a complete symbolic presentation of the transformations used to transform 3D world coordinates into 2D vertices on an image plane. Therefore, you can easily recognize and reason about the significant components in this transformation.
30. In the traditional rendering pipeline, mapping polygon vertices into the canonical view volume is only the first step in a two-step process. That second step involves painting pixels based upon those polygons in such a way that hidden surfaces are not rendered. At the highest level, this means you can distinguish between the manner in which a ray tracing algorithm is pixel driven where a rendering pipeline his polygon driven.
31. Cohen and Sutherland introduced a very elegant algorithm for quickly determining if a line segment needed to be clipped to the image plane and also where it needed to be clipped, i.e. against which side of the four-sided image plane. While arguably not in common use today, this is an elegant algorithm that you should nonetheless understand. It stands in for a myriad of more modern techniques where cleverness leads to efficiency.
32. It may be very mechanical, but being able to determine the precise endpoint of a 2D line segment resulting from clipping against the rectangular image plane it is something you can do and do quickly. Keep in mind this is a special case of the problem that arises when intersecting two parametric line segments. Consequently, it has a simpler solution.
33. The odd-even parity rule often that defines what it means for a pixel to be interior to a polygon and also represents a key aspect of how polygon filling algorithms our encoded. As part of your basic literacy and computer graphics you understand this rule and can use it to answer questions about whether an example is or is not correctly filled.
34. You can be grateful that you are almost certainly never going to be called upon to write a scan conversion algorithm that fills pixels given the vertices of a polygon. That said, understanding scan conversion is important if for no other reason that it represents one of those places in computer graphics where an understanding of actual pixel geometry along with the interaction of multiple polygons becomes important. It is also important insomuch as processes like rendering with a Z buffer don't make sense unless you first understand what rendering looks like at the pixel level.
35. Given that we covered polygon filling the semester, you should be able to answer questions about which pixels are in which pixels are not filled following the basic algorithm introduced in lecture 18.
36. in lecture 19 the extreme nonlinearity of the pseudo-depth value generated when moving into the canonical view volume was explained in detail. Consequently, you're comfortable with that nonlinearity, you're also comfortable with why it does not matter if the goal is simply to determine which of two points in the canonical view volume is closest to the camera. Finally, you're comfortable with how the actual depth could be calculated providing there was motivation to pay the extra computational charge.
37. In modern realizations of the rendering pipeline actual code is written to shade individual vertices and even pixels. Unfortunately, the complexity in degrees of freedom open to the programmer preclude our studying these shaders in any detail. However, prior to the introduction of shaders, shading could be counted upon two arise from a hand full of well understood algorithms. Three that you should understand well our: flat shading, Gouraud Shading and Phong Shading.
38. Interpolating intermediate values while moving across a polygon comes up again and again as a fundamental operation in shading. It also so happens that in the context of interpolating surface normals for ray tracing the processes is nearly trivial given what we already know about our Ray Triangle intersection code. Be sure you understand this point and can explain it.