Introduction to Graphics
Exercise Sheet

Andrej Ivašković (ai294)
Compiled on: 14th November 2019

Some of the exercises have been borrowed or adapted from question in old *Computer Graphics and Image Processing* exercise sheets. Credit for those is due to their authors.

This exercise sheet only contains theoretical and conceptual exercises – writing OpenGL and GLSL code is covered in detail in the ticks.

1 Mathematical foundations

1.1 Before attempting the problems

These exercises cover lectures 1–4: image representation, the basic ray tracing algorithm, as well as an overview of the graphics pipeline and different coordinate systems used in computer graphics. Note how 4-vectors and matrix multiplication are ubiquitous here: if you are uncomfortable with these ideas, it might be a good idea to review them.

To get a feel of the work done by 3d artists, download Blender. Its user interface is not the most intuitive, but you will see that you will see that 3d artists use the same language and ideas as OpenGL programmers and computer scientists. Try importing or exporting some `.obj` files, then open them in a text editor. You should be able to see how 3d objects are represented in a format that is easy to parse both by humans and by programs.

1.2 Problems

Exercise 1.1. Ray tracing and the rendering pipeline (somewhat resembling the OpenGL pipeline) are two methods for creating an image based on a scene. In which scenarios would you use one over the other?

Exercise 1.2.

(*a*) Explain the concepts of *diffuse* and *specular reflection*.

(*b*) Why do we care about approximating the specular component as opposed to performing the exact computation?

If the intensity of the incoming ray is I_{in}, the proportion of light specularly reflected is k_{spec}, the unit vector of the normal is $\vec{n}$, the unit vector of the incoming ray in $\vec{r}_{\text{in}}$, the unit vector of the direction of perfect reflection is $\vec{r}_{\text{perf}}$, and the unit vector pointing towards the viewer is $\vec{v}$, what is the formula for computing I, the intensity of the specular component of reflected light in the direction of the viewer?

If n is Phong's roughness coefficient, what is Phong's approximation of specular reflection? Show that these two expressions are equal when $n \to \infty$.

(*c*) Typically, surfaces have different types of components and their reflectance behaviour may vary in different parts – this is specified with *reflectance maps*. Reflectance maps contain the values of s and n in the following expression for $\phi(i, e, g)$, for different parts of the surface:

$$\phi(i, e, g) = s\alpha(2 \cos i \cos e - \cos g)^n + (1 - s)\beta \cos i$$

Here $\phi(i, e, g)$ specifies the fraction of incident light reflected per unit surface area, per unit solid angle, in the direction of the camera. The angle i is the angle of the light source, relative to the surface normal $\vec{n}$, e is the angle of a ray of light re-emitted from the surface, also relative to $\vec{n}$, and g is the angle between the emitted ray the light source. In addition, α and β are some positive real constants.

Show mathematically how this formula corresponds to a combination of specular and diffuse reflection.

Hint: express the angle cosines in terms of unit vectors.

Exercise 1.3.

(*a*) Describe in your own words what *aliasing* is (in the context of computer graphics). Your answer should cover both *spatial* and *temporal* aliasing.

(*b*) Explain *three* methods of anti-aliasing in ray tracing. Describe their strengths and weaknesses.

(*c*) Is there ever need to perform anti-aliasing in the rendering pipeline?

Exercise 1.4. What is *distributed ray tracing*?

Exercise 1.5. Describe the steps of the coordinate transformation pipeline. Explain the purpose of every representation and transition, and what kind of coordinates are used in which step.

Exercise 1.6. Explain why matrix operations are ubiquitous in 3d graphics rendering. If everything is happening in three dimensions, why don't we stick to 3-vectors and 3×3 matrices?

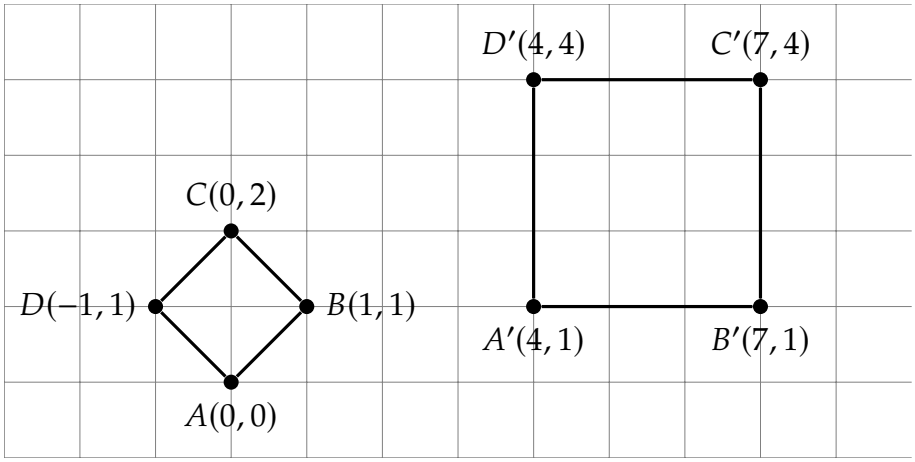

Figure 1

Exercise 1.7. *(2010 Paper 4 Question 4, (b))* Give a matrix that transforms the square $ABCD$ into $A'B'C'D'$ (see figure 1).

Exercise 1.8. Given a finite set of weighted points $S = \{(A_1, m_1), \ldots, (A_n, m_n)\}$, where the masses are real numbers such that $\sum_{i=1}^{n} m_i \neq 0$, the *centre of mass* of S is the unique point C such that, for all points O in space:

$$\overrightarrow{OC} = \frac{\sum_{i=1}^{n} m_i \overrightarrow{OA_i}}{\sum_{i=1}^{n} m_i}$$

Given a triangle $\triangle ABC$ and a point P in its plane, its *barycentric coordinates* relative to $\triangle ABC$ is the tuple (m_A, m_B, m_C) such that $m_A + m_B + m_C = 1$ and P is the centre of mass of $\{(A, m_A), (B, m_B), (C, m_C)\}$.

(*a*) How can you compute the barycentric coordinates of a point P inside $\triangle ABC$? Derive your answer.

(*b*) How can barycentric coordinates be used to implement Gouraud and Phong shading?

Note that the uniqueness of barycentric coordinates and the centre of mass can be proved with a bit of vector manipulation – do so if you want to review vector operations.

2 OpenGL and colour

2.1 Before attempting the problems

This set of exercises concerns the latter half of the course: the details of the rendering pipeline in OpenGL, textures, practical rendering issues, fundamentals

of colour theory. Once again, this exercise sheet does not contain any coding assignments.

A nice thing about this course is that every problem you come across can be visualised and intuitively understood. The solutions themselves are varied and there are many more than are presented in the course (the history of development of these techniques is really interesting). The mathematical background and linear algebra require developing some intuition – hopefully this will be provided to you through Maths courses.

2.2 Problems

Exercise 2.1.

(*a*) What are the advantages of GPUs over CPUs and why are they better suited for graphics rendering?

(*b*) What is the difference between OpenGL and GLSL?

(*c*) Give a brief high-level overview of the OpenGL rendering pipeline and all of its steps.

(*d*) Are the front and back clipping planes of a *viewing frustrum* both necessary?

(*e*) In what step is the *z-buffer* algorithm relevant?

Exercise 2.2.

(*a*) What is the difference between *bump* and *displacement maps*? Why would you use one over the other?

(*b*) Explain the purpose of *MIP maps*.

(*c*) How can you use textures to render the sky and clouds?

Exercise 2.3. Find a UV map for projecting a texture onto a unit sphere centred at the origin. Are there any mathematical and practical issues with doing this?

Exercise 2.4. *Decimation* is a is a process of reducing the number of polygons of a model, while still remaining sufficiently similar to the original.[1]

In 3d animated films and especially in video games, it is common to have several 3d models for the same entity with different *levels of detail* (LOD).[2] For characters these will usually be created using a more manual approach, whereas for environment art (especially terrain) it is more common to make heavy use of automated decimation tools. Typically you would have at least three or four LODs for terrain.

[1] In Blender this is done by adding a Modifier to an Object.

[2] Versions 4.x of OpenGL add a *Tessellation* shader between the vertex and fragment shaders, which can be used to create different LODs – though you would start with a low-poly model and construct a high-poly one from it.

(*a*) Why might it be useful to load the same object at different LODs?

(*b*) What is the implication for textures?

(*c*) Describe a decimation algorithm that takes as input a 3d mesh formed out of n triangles and a number p, where $0 < p < 1$, and outputs a mesh made out of approximately np triangles.

Exercise 2.5. Describe the problems of *flicker* and *tearing*. Give an overview and describe the techniques used to solve these issues.

Exercise 2.6. You are implementing a graphics engine for a survival horror game. Your task is to add fog into some scenes. The model that you use is that at distance 0 from the viewer there is no change in visibility, at distance d the fog totally takes over, the colour of the fog is c, and an exponential increase in fog strength holds in between 0 and d: the fog density f ('proportion of fog') at distance x follows a law of the form:

$$f(x) = a + b\exp(\lambda x), \quad 0 < x < d$$

for a set parameter λ. Find the values of a and b and give a sketch of what needs to be done in OpenGL code (in setup and the shaders) to model this.

Exercise 2.7. Using blue lines on a black background in a CAD system would be a poor choice for the interface colours for designing an object. Why? Your answer should mention the biology of the human eye.

Exercise 2.8. Many computer vision algorithms (edge and object detection, motion tracking, inferring depth, image recognition) tend to convert RGB images into HSV at the start and work with that representation. Why might this be the case?

Exercise 2.9.

(*a*) What is *metamerism*?

(*b*) What is the difference between *luma* and *luminance*?

(*c*) Why does *sigmoidal tone mapping* make sense?

(*d*) What is the motivation for adding *glare* effects?

(*e*) Explain *gamma correction*.