# Faculty of Science
# COMP-557 - Fundamentals of Computer Graphics (Fall 2020)
# Midterm Examination - Version 1

Available on MyCourses Thursday 8 October 2020, 1:00 PM EDT
Submit on MyCourses before Sunday 11 October 2020 1:00 PM EDT
Examiner: Prof. Paul Kry

## Instructions:

- Read the entire exam before starting to write any answers!
- This is an "in class" exam, i.e., budget 90 minutes within the 72 hour window to complete the exam.
- This is an open book exam; you may use your notes, the slides, and textbook.
- You will not need and, should not use any other resources (e.g., google search).
- Working in groups and discussing problems is strictly prohibited!
- All your answers must be your own.
- If you write your answers on paper, ensure that you write clearly and your scan has good contrast.
- Submit your written exam as a PDF on MyCourses.
- Marks associated with each written answer are shown in the left margin.
- There is a total of 20 marks.
- Read questions carefully.

**[2]** 1. Given a 3D point $\mathbf{p} \in \mathbb{R}^3$ and a unit length vector $\mathbf{z} \in \mathbb{R}^3$, write pseudocode for a robust (i.e., never fails) algorithm that constructs and returns an orthonormal coordinate frame.

*That is, your function should produce the homogeneous 4-by-4 matrix that maps points and vectors in coordinates of this new frame to points and vectors in the coordinates of the world frame.*

**[2]** 2. Describe how one can use a **similarity transform** to implement a rotation about an arbitrary axis by a given angle.

*Please note that you should not work out the matrix product.*

**[2]** 3. Design a 4x4 perspective projection matrix that works much like the one we saw in class (i.e., the $x$ and $y$ components of a point are scaled by their distance along the $z$ axis), except that all points are also projected onto the near plane. Let $n$ be the distance of the near plane along the negative $z$-axis.

**[3]** 4. Given your matrix $P \in \mathbb{R}^{4\times4}$ from Question 3, and the change of frame construction in Question 1, describe how to create a **similarity transform** that will produce cheap shadow projections for a point light at location $\mathbf{a} \in \mathbb{R}^3$ casting "cheap" shadows onto a plane with normal $\mathbf{n} \in \mathbb{R}^3$ that contains the point $\mathbf{b} \in \mathbb{R}^3$. Show your work and draw a picture to help explain your answer.

*Please note that you should not work out the matrix product. Also note that your answer with a similarity transformation will be simpler than the approach on slide 17 of the ObjectOrderShadows slides from class!*

**[2]** 5. Consider a scene where your camera is looking at a wall 2 m away and you draw a quadrilateral in front at 1 m distance along the negative z axis. To draw this scene we need to choose near and far clipping planes and be careful to not loose precision of the z coordinates after projection (to distinguish poster fragments from wall fragments). Suppose the far plane is at a distance of 10 m, and that $\epsilon$ is the smallest discernible distance in depth, post projection. What is the closest near plane you can set?

*Hint: compute $n$ for when the difference of depths post projection is equal to $\epsilon$.*

**[4]** 6. Consider a scene where your camera is looking a sphere of radius 1 m. Let the center of the sphere be at distance of 2 m on the negative $z$ axis of the camera frame. Suppose there is a ceiling above at a height of 3 m in the $y$ direction of the camera frame. Suppose want the brightest spot of a Blinn-Phong specular highlight at a point which is centered horizontally, and at a height of 0.6 m above the center of the sphere (i.e., on the sphere, not in the image of the sphere). Where should a point light be placed on the ceiling to produce this highlight?

*Hint: first find identify the point, then the direction of the viewer, then the direction of the light, and then finally find its position by intersecting a parametric line with the plane equation for the ceiling.*

**[3]** 7. On the next page is a GLSL vertex program and fragment program for doing Lambertian and Blinn-Phong illumination with a point light. The code has many elements of a correct solution, but also has a number of important bugs! List the problems you find and the line numbers along with simple fix for each bug.

**[2]** 8. Did you attended Ken Museth's SCA 2020 keynote? If yes, describe what you found most interesting in his presentation.

```
-------------------------- Vertex Shader ----------------------------
#version 400 core
uniform mat4 M;     // modeling matrix
uniform mat4 V;     // view matrix
uniform mat4 P;     // projection matrix
uniform mat3 MinvT; // inverse transpose of linear part of M
uniform mat3 VinvT; // inverse transpose of linear part of V

in vec3 VertexNormal;
in vec4 VertexPosition;

out vec4 PositionForFP; // camera coordinates
out vec3 NormalForFP;   // interpolate the normalized surface normal

void main() {
    NormalForFP = MinvT * VinvT * VertexNormal;
    PositionForFP = V * M * VertexPosition;
    gl_Position = P * V * M * VertexPosition;
}
```

```
------------------------- Fragment Shader ---------------------------
#version 400 core
uniform vec3 LightColor;     // rgb intensity of light
uniform vec3 LightPosition;  // light position in camera coordiantes
uniform float Shininess;     // exponent for Blinn-Phong model
uniform vec3 kd;             // diffuse material property

in vec4 PositionForFP; // fragment position in camera coordinates
in vec3 NormalForFP;   // normal at each fragment in camera coordinates

out vec4 FragColor;

void main() {
    vec3 LightDirection = PositionForFP - LightPosition;
    float diffuse = dot( NormalForFP, LightDirection );
    vec3 ViewDirection = vec3(0,0,0) - PositionForFP;
    HalfVector = (LightDirection + ViewDirection) / 2;
    float specular = max(0.0, dot(NormalForFP, HalfVector));
    if (diffuse == 0.0) {
        specular = 0.0;
    } else {
        specular = pow( specular, Shininess );
    }
    vec3 scatteredLight = kd * LightColor * diffuse;
    vec3 reflectedLight = LightColor * specular;
    vec3 rgb = min( scatteredLight + reflectedLight, vec3(1,1,1) );
    FragColor = vec4( rgb, 1 );
}
```