Last Name: _____ First Name: _____ Cornell NetID: _____

# CS 4620 Midterm, October 23, 2018

# SOLUTION

1. [20 points] **Transformations**

   (a) Describe the action of each of the following matrices, as transformations in homogeneous coordinates, in terms of rotation, scaling, and/or translation. For instance, the answer to (0) is "a scale by a factor of 2 in the $x$ direction followed by a translation of 1 in $x$."

$$(0): \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1): \begin{bmatrix} \cos(30°) & 0 & \sin(30°) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(30°) & 0 & \cos(30°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2): \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad (3): \begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

   (1)

   (2)

   (3)

   (b) Write each of the matrices (2) and (3) from part (a) as a product of three matrices in the form $TXT^{-1}$ where $T$ is a translation and $X$ is a rotation or scale about the origin.

   (2)

   (3)

   (c) Describe each of the matrices (2) and (3) from part (a) as a rotation or scale about a point. For instance, one answer to (0) would be "a scale by a factor of 2 in the $x$ direction about the point $(-1, 0, 0)$."

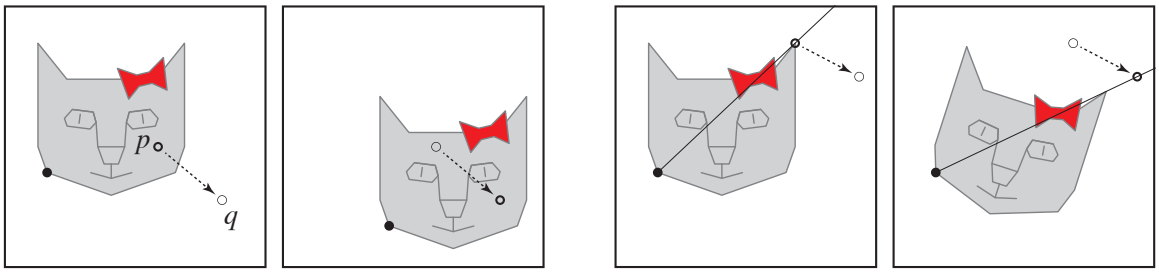
   (2)

(3)

2. [20 points] **Manipulation**

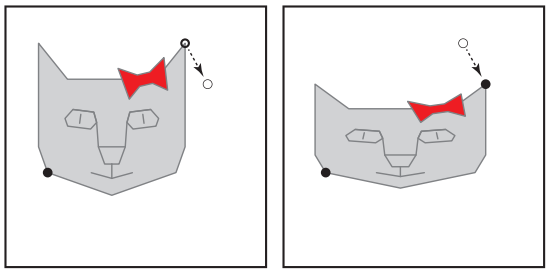
Suppose you are implementing a 2D manipulation interface for a touchscreen device. Like our Manipulators assignment, the UI has three modes, for translation, rotation, and scaling, and it stores the object's transformation as a product $TRS$ where $T$, $R$, and $S$ are a translation, a rotation about the origin, and a scale about the origin.

But things are simpler than in the assignment; since the input and the manipulated object are both 2D, there is no need for separate manipulation axes. Instead, when the user touches at $p$ and moves to $q$, we want to apply a translation, a rotation about the origin, or a scale about the origin in the object's local coordinates, that will make the point that appears on the screen at $p$ move to $q$ (or comes as close to $q$ as possible, in the case of rotation).




Translate mode

Rotate mode



Scale mode

This is a 2D object, so the view transformation is a 2D affine transformation $M_V$ that maps the visible area of the object to the rectangle $[-1, 1] \times [-1, 1]$; we'll call this 2D space NDC (normalized device coordinates) just like in 3D. The view transformation is fixed during the manipulation process. (Probably the UI provides some way for the user to scroll and zoom, which would change $M_V$, but we're not concerned with that here.)

(a) What point in object space transforms to the point $p$ in NDC?

(b) Given the points $p$ and $q$ in NDC, work out the object's new transformation after a touch interaction in each of the three modes, in terms of the matrices $T$, $R$, and $S$ that make up its object transformation before the interaction. You can use the notation $T(v)$ for a translation by the vector $v$, $R(\theta)$ for a rotaion by the angle $\theta$, and $S(a, b)$ for a scale by $a$ on the $x$ axis and $b$ on the $y$ axis.

For instance, your answer to (1) will be in the form $TT(v)RS$ where $v$ is some vector (you have to figure out what it is) that depends on the points $p$ and $q$.

(1) in translation mode, what is the object's transformation after a drag from $p$ to $q$?

(2) in rotation mode, what is the object's transformation after a drag from $p$ to $q$?

(3) in scaling mode, what is the object's transformation after a drag from $p$ to $q$?

Hint: you can check all three answers by ensuring that the point that transforms to $p$ under $TRS$ transforms to $q$ under your updated transformation.

Solution:

- 1(a)
  - (1) A counter-clockwise rotation about y axis by 30 degrees.
  - (2) A counter-clockwise rotation by 90 degrees in 2D plane and then translate along y axis by 1.
  - (3) A scale about the origin by a factor of 2 in both x and y directions; then translations along both x and y directions by -1.
- 1(b)

$$(2): T = \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3) T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 1(c)
  (2) Counter-clockwise rotation about (-0.5,0.5) for 90 degrees;
  (3) Scale about point (1,1) for factor of 2 in both x and y directions.

- 2(a)

$$S^{-1}R^{-1}T^{-1}M_V^{-1}p$$

- 2(b)
  - (1) $TT(v)RS$ where $v = M_V^{-1}q - M_V^{-1}p$. ($T^{-1}M_V^{-1}q - T^{-1}M_V^{-1}p$ also OK).
  - (2) $TRR(\theta)S$ where $\theta$ is the angle between $p' = T^{-1}M_V^{-1}p$ and $q' = T^{-1}M_V^{-1}q$. Theta can be computed various ways; for example $\theta = \text{atan2}(q'_y, q'_x) - \text{atan2}(p'_y, p'_x)$ or $\theta = \sin^{-1}(w_z)$ where $w = [\hat{p}'; 0] \times [\hat{q}'; 0]$. The solution $\theta = \cos^{-1}(\hat{p}' \cdot \hat{q}')$ requires an additional check for the sign of $\theta$. ($p' = R^{-1}T^{-1}M_V^{-1}p$ and $q' = R^{-1}T^{-1}M_V^{-1}q$ also OK.)
  - (3) $TRSS(a, b)$ where $a$ and $b$ are the ratios of $x$ and $y$ coordinates between points $p' = R^{-1}T^{-1}M_V^{-1}p$ and $q' = R^{-1}T^{-1}M_V^{-1}q$. Specifically $a = q'_x/p'_x$ and $b = q'_y/p'_y$. ($p' = S^{-1}R^{-1}T^{-1}M_V^{-1}p$ and $q' = S^{-1}R^{-1}T^{-1}M_V^{-1}q$ also OK.)

3. [20 points] **Geometry images**

In displacement mapping we use a scalar texture to modify the vertex positions of a mesh. In some cases the input geometry can be pretty boring (like a flat square) and everything interesting about the shape comes from the displacement texture (e.g. a mountain range made by using an elevation map for the displacement). A related technique to displacement mapping is *geometry images*, which takes this one step farther: the original vertex position is completely ignored, and the actual position is completely determined by a 3D *position texture*.

In this problem we want you to write a **vertex shader** for geometry images. Specifically, your shader should expect vertices with texture coordinates as the only vertex attribute, along with the same uniform matrices we used in the Shaders assignment (see declarations below). The object-space position of each vertex is defined by the value of the position texture at the corresponding uv coordinates. The fragment shader is provided and is completely standard: it uses the position and a varying normal to compute simple Lambertian lighting.

Hints: don't forget you will need to compute the normal. The computation of the normal uses exactly the same ideas as in displacement mapping, but is much simpler. You don't need the varyings derivU and derivV from the assignment, but you do still need to differentiate the texture. You can use exactly the same approach as in the assignment (a finite difference with a fixed stepsize of 0.0001), and don't be concerned about any discretization or quantization issues for this problem.

```
//  --- begin vertex shader ---
// = object.matrixWorld
uniform mat4 modelMatrix;

// = camera.matrixWorldInverse * object.matrixWorld
uniform mat4 modelViewMatrix;

// = camera.projectionMatrix
uniform mat4 projectionMatrix;

// = camera.matrixWorldInverse
uniform mat4 viewMatrix;

// = inverse transpose of modelViewMatrix
uniform mat3 normalMatrix;

// = camera position in world space
uniform vec3 cameraPosition;

attribute vec2 uv;

varying vec3 vNormal;
```

```
varying vec4 vPosition;

uniform sampler2D positionTexture;

main() {
        // TODO: write your code here









}

//--- end vertex shader ---

//--- begin fragment shader ---
uniform vec3 lightColors[ NUM_LIGHTS ];
uniform vec3 lightPositions[ NUM_LIGHTS ]; // in eye space

uniform float exposure;
uniform vec3 diffuseColor;

varying vec3 vNormal;
varying vec4 vPosition;

vec3 to_sRGB(vec3 c) { return pow(c, vec3(1.0/2.2)); }
vec3 from_sRGB(vec3 c) { return pow(c, vec3(2.2)); }
```

```
void main() {

    vec3 N = normalize(vNormal);
    vec3 V = normalize(-vPosition.xyz);

    vec3 finalColor = vec3(0.0, 0.0, 0.0);

    for (int i = 0; i < NUM_LIGHTS; i++) {
        float r = length(lightPositions[i] - vPosition.xyz);
        vec3 L = normalize(lightPositions[i] - vPosition.xyz);
        vec3 H = normalize(L + V);

        // calculate diffuse term
        vec3 Idiff = diffuseColor * max(dot(N, L), 0.0);
        finalColor += lightColors[i] * Idiff / (r*r);
    }

    // Only shade if facing the light
    if (gl_FrontFacing) {
        gl_FragColor = vec4(to_sRGB(finalColor * exposure), 1.0);
    } else {
        gl_FragColor = vec4(170.0/255.0, 160.0/255.0, 0.0, 1.0);
    }
}

//--- end fragment shader ---
```

Solution:

```
    // In vertex shader

main() {
    vec3 pos = texture2D(positionTexture, uv).xyz; // 2 pts
    vPosition = modelViewMatrix * vec4(pos, 1.0); // 3 pts

    // 4 pts
    float du = 0.001;
    float dv = 0.001;
    vec2 duVector = vec2(du, 0.0);
    vec2 dvVector = vec2(0.0, dv);
    vec3 posU1 = texture2D(positionTexture, uv-duVector).xyz;
    vec3 posU2 = texture2D(positionTexture, uv+duVector).xyz;
    vec3 posV1 = texture2D(positionTexture, uv-dvVector).xyz;
    vec3 posV2 = texture2D(positionTexture, uv+dvVector).xyz;

    // 3 pts
    vec3 dpdu = (posU2 - posU1)/du/2.0;
    vec3 dpdv = (posV2 - posV1)/dv/2.0;

    vec3 normal = normalize(cross(dpdu, dpdv)); // 3 pts
    vNormal = normalize(normalMatrix * normal); // 3 pts

    gl_Position = projectionMatrix * vPosition; // 2 pts
}
```

4. [20 points] **True/False**

Circle whether each statement is true (T) or false (F). If it is true, explain why. If it is false, explain why or provide a counterexample.

(a) T F An indexed triangle mesh will generally consume more storage than the same mesh written as a list of separate triangles.

(b) T F The normal vector used for shading on a triangle mesh is always perpendicular to the surface being shaded.

(c) T F In barycentric coordinates, the coordinates of a triangle's vertices are all zeros and ones.

(d) T F Storing one additional byte at every vertex of an indexed triangle mesh generally uses more space than storing one additional byte for every triangle of the same mesh.

(e) T F The intersection between a ray and a sphere is generally found using an implicit model of the sphere.

(f) T F In orthographic viewing, the viewing rays all share the same ray direction.

(g) T F If $A$ and $B$ are two $2 \times 2$ rotation matrices, then $AB = BA$.

(h) T F If a $4 \times 4$ matrix represents a 3D affine transformation in homogeneous coordinates, then its last row must be $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$.

(i) T F If a projective transformation is applied to a set of points evenly spaced along a line, the resulting points will be collinear and evenly spaced.

(j) T F In homogeneous coordinates, the 3D point $(x, y, z)$ is represented by the 4D point $(x, y, z, 0)$.

Solution:

(a) False. When stored as a list of separate triangles, each vertex is stored once for each triangle of which it is a member, which is twice on average. When stored as an indexed mesh, each vertex is stored only once. The total storage is on average 72 bytes per vertex when stored as a list of separate triangles compared to 36 bytes per vertex in an indexed mesh.

(b) False. If this were the case, then all triangle meshes would appear to be faceted. Averaging normals creates a smoother appearance by using normals that are not perpendicular to any of their adjacent triangles.

(c) True. The coordinates of a triangle's vertices in barycentric coordinates are $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. The coordinates represent linear combinations of the vertices' positions.

(d) False. There are on average two triangles per vertex in a mesh. When using an indexed mesh, the statement would be false because vertices are stored once. The statement would be true if the mesh is not indexed: then each vertex is stored on average 6 times, so one additional byte per vertex would generally use more space.

(e) True. Storing a sphere as an implicit model allows an algebraic solution for $t$ when intersecting the sphere with a ray and results in a smooth sphere appearance.

(f) True. This is the definition of orthographic viewing.

(g) True. The rotations are around the origin, so it does not matter in which order they are applied.

(h) True. When not using projective transformations, homogeneous coordinates allow for compact representation of affine transformations, including translations. The final row then serves to copy over the $w = 1$ term.

(i) False. Perspective projection provides a counterexample. The denominator of $y$ varies with $z$, so distance is not preserved: distances that are farther away will appear to be shorter than equivalent distances that are closer. The statement is true for affine transformations.

(j) False. $(x, y, z)$ is represented by $(x, y, z, 1)$ in homogeneous coordinates. $(x, y, z, 0)$ represents a point at infinity in the direction of $(x, y, z)$.

Last Name: _____ First Name: _____ Cornell NetID: _____

5. [20 points] **Meshes**

Consider four OBJ files, all with the same vertex data but indexed in different ways.

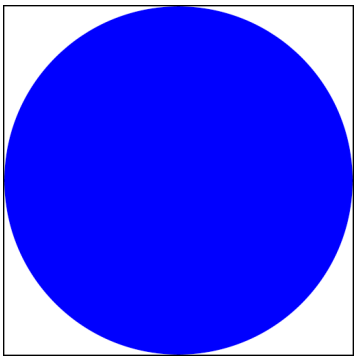
(a) Which of these will produce faceted shading?

(b) Which has continuous texture coordinates?

(c) For each of these 4 meshes, match them with the corresponding rendering on the next page.

```
--- vertex data ---
v 1 1 1
v 0 1 1
v 1 0 1
v 1 1 0
vn 1 1 1
vn 1 0 0
vn 0 1 0
vn 0 0 1
vn 0 1 1
vn 1 0 1
vn 1 1 0
vt 0 0
vt 0 1
vt 1 0
vt 0.75 0.75
vt 0.5 0.5

--- index data 1 ---
f 1/5/4 2/5/4 3/5/4
f 1/5/2 3/5/2 4/5/2
f 1/5/3 4/5/3 2/5/3

--- index data 2 ---
f 1/1/4 2/2/4 3/3/4
f 1/1/2 3/2/2 4/3/2
f 1/1/3 4/2/3 2/3/3

--- index data 3 ---
f 1/1/1 2/2/5 3/3/6
f 1/1/1 3/2/6 4/3/7
f 1/1/1 4/2/7 2/3/5

--- index data 4 ---
f 1/4/1 2/1/5 3/2/6
f 1/4/1 3/2/6 4/3/7
f 1/4/1 4/3/7 2/1/5
```

Texture File:



Hint: OBJ indices start at 1, and the ordering of index data is position/texture/normal.

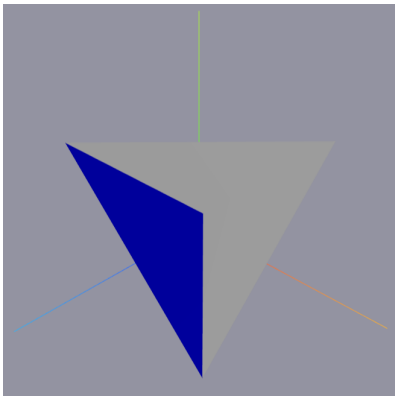
| Has Faceted Shading? | Has Continuous Texture Coords? | Rendering Number |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Last Name: _____ First Name: _____ Cornell NetID: _____

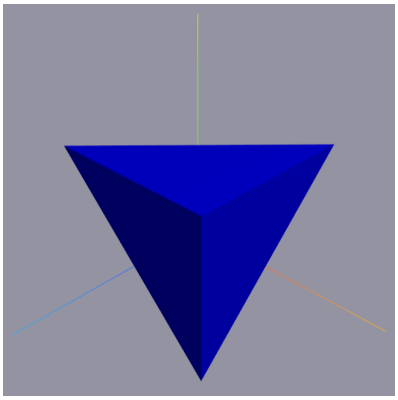
Solution:

| Has Faceted Shading? | Has Continuous Texture Coords? | Rendering Number |
|---|---|---|
| Yes | Yes | 2 |
| Yes | No | 7 |
| No | No | 8 |
| No | Yes | 3 |

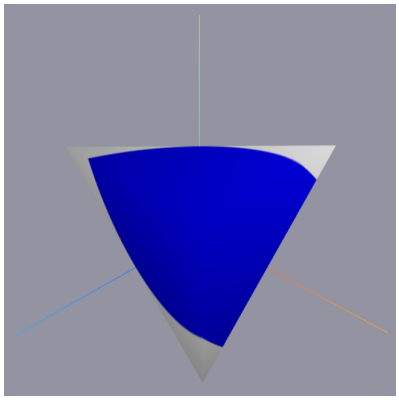
Last Name: _____ First Name: _____ Cornell NetID: _____



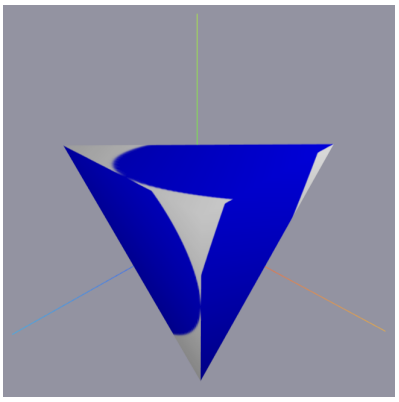
(1)



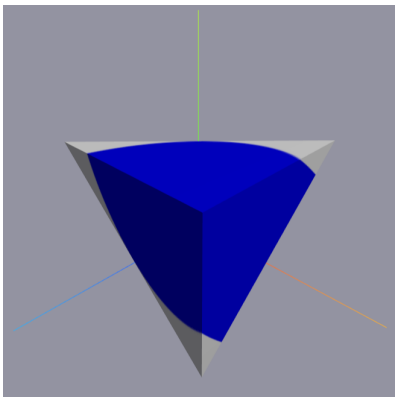
(2)



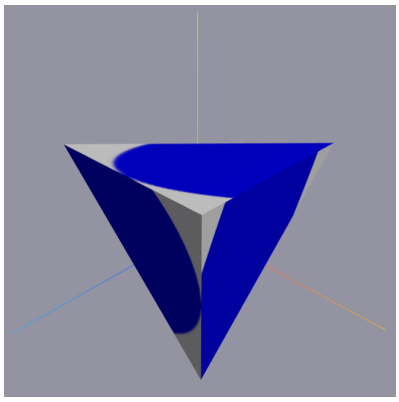
(3)



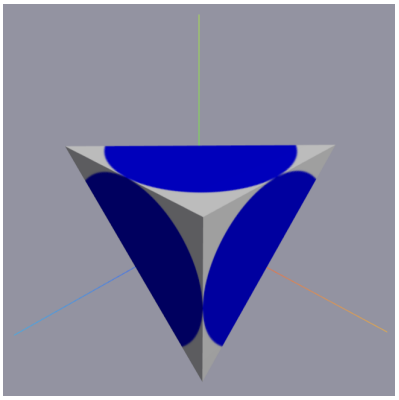
(4)



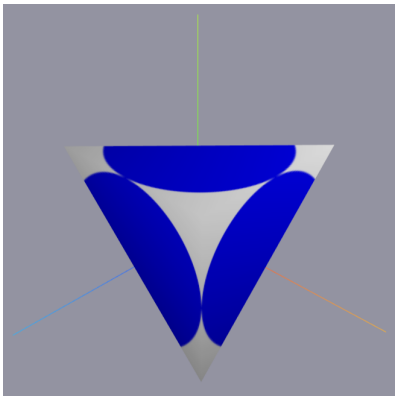
(5)



(6)



(7)



(8)