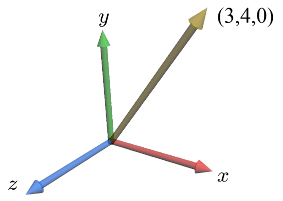
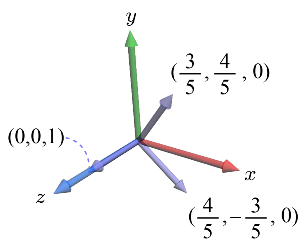
Chapter 2

1. Construct a 3D orthonormal basis, where the first basis vector is along $(3, 4, 0)$, the second is along a principal axis, and the last is obtained by taking their cross product.



$\{(3/5, 4/5, 0), (0, 0, 1), (4/5, -3/5, 0)\}$



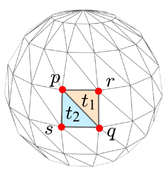
y
$(\frac{3}{5}, \frac{4}{5}, 0)$
(0,0,1)
z
x
$(\frac{4}{5}, -\frac{3}{5}, 0)$

2. A line segment is defined by two end points, p_0 and p_1, where $p_0 = (2, 0)$ and $p_1 = (5, 0)$. Suppose that the vector assigned to p_0 is $(-1, 2)$ and that assigned to p_1 is $(2, 5)$. On the line segment, consider a point at $(4, 0)$. Compute the vector at the point by linearly interpolating the vectors at p_0 and p_1.

$(1, 4)$

Chapter 3

1. Consider the two triangles in the figure. Making sure that the triangle normals point out of the object, fill in the vertex and index arrays for the indexed mesh representation.



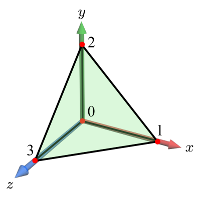
vertex array

0	p
1	q
2	r
3	s

index array

0	0
1	1
2	2
3	3
4	1
5	0

2. The simplest 3D closed mesh is a *tetrahedron*. Suppose that it is composed of four vertices, $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. Making sure that the triangle normals point out of the tetrahedron, draw the vertex and index arrays for the indexed mesh representation.



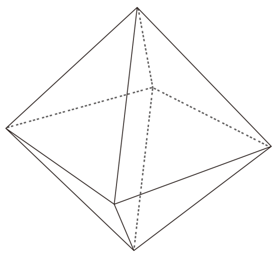
vertex array

0	(0,0,0)
1	(1,0,0)
2	(0,1,0)
3	(0,0,1)

index array

0
2
1
1
2
3
3
2
0
3
0
1

3. The triangle mesh shown below is composed of eight triangles. In the non-indexed mesh representation, the vertex array has (   ) elements. In the indexed representation, the vertex and index arrays have (   ) and (   ) elements, respectively. Fill in each blank by a number.

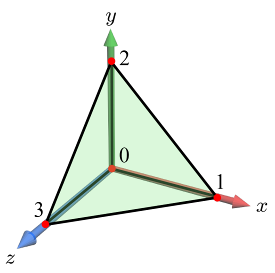


24, 6, 24

4. Consider a sphere, the center of which is located at the origin. Let us create a polygon mesh for the sphere by defining vertices at every 10 degrees along both latitude and longitude. How many vertices do we have?

614 vertices in total.

5. Given the triangle mesh shown below, fill in the vertex and index arrays for its indexed mesh representation.



	vertex array
0	(0,0,0)
1	(1,0,0)
2	(0,1,0)
3	(0,0,1)

index array
0
2
1
0
1
3
3
2
0
1
2
3

6. Let v, e, and f denote the numbers of vertices, edges and faces of a closed triangle mesh, respectively. In [Note: Vertex-triangle ratio in a triangle mesh], we have derived $f = 2v - 4$. Derive a similar relation between v and e.

Then, the relation between v and e is as follows: $e = 3v - 6$

Chapter 4

1. Note the difference between a column vector and a row vector. For matrix-vector multiplication, let us use row vectors.

   (a) Write the translation matrix that translates (x, y) by (dx, dy).

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix}$$

   (b) Write the rotation matrix that rotates (x, y) by θ.

$$\begin{pmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

   (c) Write the scaling matrix with scaling factors s_x and s_y.

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2. The object-space basis of an object is denoted by $\{u, v, n\}$. Initially, it is the same as the world-space basis. Suppose that $u = (0, 0, -1)$ and $n = (1/\sqrt{2}, 1/\sqrt{2}, 0)$ after a rotation. Compute the matrix for the inverse of the rotation.

$$\begin{pmatrix} 0 & 0 & -1 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{pmatrix}$$

3. The following matrix represents a combination of a rotation, R, followed by a translation, T. Write the matrices for R and T.

$$\begin{pmatrix} -1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

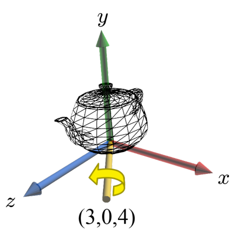
$$R = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Describe the general procedure for rotating about an arbitrary axis by θ. The procedure should include (1) the step for transforming the arbitrary axis into a principal axis and (2) the cross-product operation.

   Hint: Considering an orthonormal basis, {u, v, n}, modify the given vector so that it can be taken as either u, v, or n.

5. The teapot is to be rotated about $(3, 0, 4)$ by $90°$. The rotation about such an arbitrary axis can be defined as a combination of three matrices: (1) the rotation of the arbitrary axis onto the x-axis, (2) the rotation about the x-axis by $90°$, and (3) the inverse of (1). Compute the three matrices. [Hint: You may use $R_y(\theta)$ for (1).]



$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{pmatrix}$$

The rotation for (1) is

$$R_y(\theta) = \begin{pmatrix} \frac{3}{5} & 0 & \frac{4}{5} \\ 0 & 1 & 0 \\ -\frac{4}{5} & 0 & \frac{3}{5} \end{pmatrix}$$

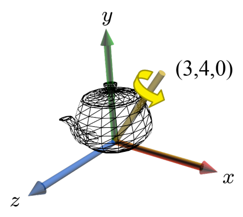
The rotation for (2) is

$$R_x(90^\circ) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

The rotation for (3) is

$$R_y(\theta)^T = \begin{pmatrix} \frac{3}{5} & 0 & -\frac{4}{5} \\ 0 & 1 & 0 \\ \frac{4}{5} & 0 & \frac{3}{5} \end{pmatrix}$$

6. The teapot is to be rotated about $(3,4,0)$ by 90°. The rotation about such an arbitrary axis can be defined as a combination of three matrices: (1) the rotation of the arbitrary axis onto the x-axis, (2) the rotation about the x-axis by 90°, and (3) the inverse of (1). Compute the three matrices. [Hint: You may use $R_z(\theta)$ for (1).]



$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The rotation for (1) is

$$R_z(\theta)^T = \begin{pmatrix} \frac{3}{5} & \frac{4}{5} & 0 \\ -\frac{4}{5} & \frac{3}{5} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The rotation for (2) is

$$R_x(90^\circ) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

The rotation for (3) is

$$R_z(\theta) = \begin{pmatrix} \frac{3}{5} & -\frac{4}{5} & 0 \\ \frac{4}{5} & \frac{3}{5} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Chapter 5

1. Given two non-standard orthonormal bases in 2D space, $\{a, b\}$ and $\{c, d\}$, compute the 2×2 matrix that converts a vector defined in terms of $\{a, b\}$ into that of $\{c, d\}$.

$$\begin{pmatrix} a \cdot c & b \cdot c \\ a \cdot d & b \cdot d \end{pmatrix}$$

2. Given two non-standard orthonormal bases in 3D space, $\{a, b, c\}$ and $\{d, e, f\}$, compute the 3×3 matrix that converts a vector defined in terms of $\{a, b, c\}$ into that of $\{d, e, f\}$.

$$\begin{pmatrix} a \cdot d & b \cdot d & c \cdot d \\ a \cdot e & b \cdot e & c \cdot e \\ a \cdot f & b \cdot f & c \cdot f \end{pmatrix}$$

3. Consider scaling along two orthonormal vectors, a and b, neither of which is identical to the standard basis vector, e_1 or e_2. The scaling factors along a and b are denoted by s_a and s_b, respectively. The scaling matrix is a combination of three 2×2 matrices. Write the three matrices.

$$\begin{pmatrix} a_x & b_x \\ a_y & b_y \end{pmatrix} \begin{pmatrix} s_a & 0 \\ 0 & s_b \end{pmatrix} \begin{pmatrix} a_x & a_y \\ b_x & b_y \end{pmatrix}$$

4. Consider scaling along three orthonormal vectors, a, b, and c, any of which is not identical to the standard basis vector, e_1, e_2, or e_3. The scaling factors along a, b, and c are denoted by s_a, s_b, and s_c, respectively. It is also observed that $a \times b = c$ where $\times$ denotes the cross product. The scaling matrix is a combination of three 3×3 matrices. Write the three matrices.

$$\begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{pmatrix} \begin{pmatrix} s_a & 0 & 0 \\ 0 & s_b & 0 \\ 0 & 0 & s_c \end{pmatrix} \begin{pmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{pmatrix}$$

5. The standard coordinate system is defined as $\{e_1, e_2, e_3, \mathbf{O}\}$, where $e_1 = (1,0,0)$, $e_2 = (0,1,0)$, $e_3 = (0,0,1)$, and $\mathbf{O} = (0,0,0)$. Consider another coordinate system named S. Its origin is at $(5,0,0)$ and basis is $\{(0,1,0), (-1,0,0), (0,0,1)\}$. Given a point defined in the standard coordinate system, compute the matrix that converts the point into S.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6. We are given the following view parameters: **EYE** $= (0, 0, -\sqrt{3})$, **AT** $= (0, 0, 0)$, and **UP** $= (0, 1, 0)$.

   (a) Write the basis and origin of the camera space.

   $n = (0, 0, -1),\ u = (-1, 0, 0),\ v = (0, 1, 0)$.
   The origin is at **EYE**.

   (b) The view transform consists of a translation and a rotation. Write their matrices.

   The translation is
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \sqrt{3} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
   and the rotation is
$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7. We are given the following view parameters: **EYE** $= (0, 0, 3)$, **AT** $= (0, 0, -1)$, and **UP** $= (-1, 0, 0)$.

   (a) Write the basis and origin of the camera space.

   $n = (0, 0, 1)$, $u = (0, 1, 0)$, and $v = (-1, 0, 0)$.
   The camera space is $\{u, v, n, \mathbf{EYE}\}$.

   (b) The view transform consists of a translation and a rotation. Write their matrices.

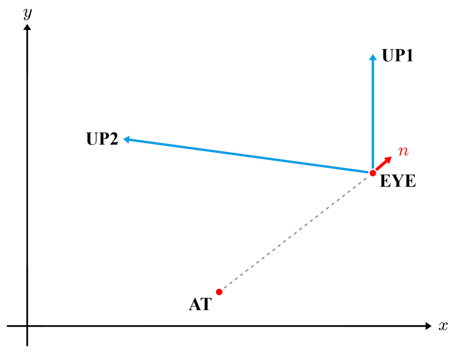
$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

8. We are given the following view parameters: **EYE** $= (0, 0, 3)$, **AT** $=$ $(0, 0, -1)$, and **UP** $= (-1, 0, 0)$. Compute the matrix that transforms the camera-space coordinates into the world space. Notice that this is not the view matrix but its inverse.

$$TR = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

9. In the world space, two different sets of view parameters are given, {**EYE**, **AT**, **UP1**} and {**EYE**, **AT**, **UP2**}, where **EYE** $= (18, 8, 0)$, **AT** $=$ $(10, 2, 0)$, **UP1** $= (0, 8, 0)$, and **UP2** $= (-13, 2, 0)$. Discuss whether the resulting camera spaces are identical to each other or not.



Two camera spaces are identical.

10. We have two camera spaces, S_1 and S_2, and want to transform a point defined in S_1 into that in S_2. A solution is to transform from S_1 to the world space and then transform from the world space to S_2.

   (a) S_1 is defined using the following parameters: **EYE** $= (0, 0, 3)$, **AT** $=$ $(0, 0, -1)$, and **UP** $= (-1, 0, 0)$. Compute the matrix that transforms a point in S_1 to that in the world space.

$$M_1 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) S_2 is defined using the following parameters: **EYE** $= (0, 0, -3)$, **AT** $= (0, 0, 0)$, and **UP** $= (0, 1, 0)$. Compute the matrix that transforms a point in the world space to that in S_2.

$$M_2 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

11. Suppose that the world space is left-handed. The view parameters are defined as follows: **EYE** $= (0, 0, 3)$, **AT** $= (0, 0, -1)$, and **UP** $= (-1, 0, 0)$.

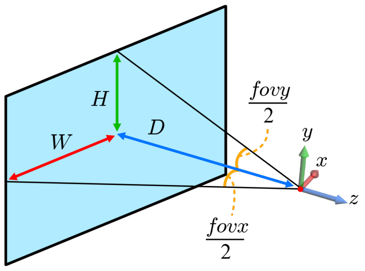
(a) Assuming that the camera space is also left-handed, compute its basis, $\{u, v, n\}$.

$u = (0, 1, 0)$, $v = (-1, 0, 0)$, and $n = (0, 0, 1)$.

(b) Compute the view matrix that converts the world-space objects into the camera space.

$$M_{view} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

12. Shown below is a cross section of the view frustum. It is orthogonal to the z-axis. We are given $\{fovy, fovx, n, f\}$, where $fovx$ stands for field of view along x-axis. D denotes the distance from the camera to the center of the cross section. Define $aspect$ as a function of $fovx$ and $fovy$.

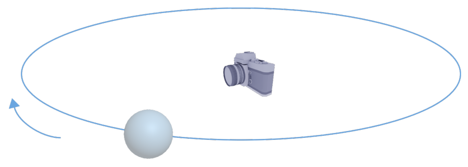


$$aspect = \frac{cot\frac{fovy}{2}}{cot\frac{fovx}{2}}$$

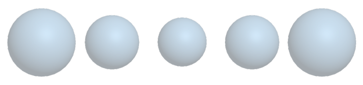
13. Section 5.4.3 derives the projection matrix based on the fact that the z-range of the cuboidal view volume is $[-1, 1]$. Assume that the z-range of the view volume is changed to $[-1, 0]$ whereas the x- and y-ranges remain $[-1, 1]$. Derive the new projection matrix.

$$\begin{pmatrix} \frac{cot\frac{fovy}{2}}{aspect} & 0 & 0 & 0 \\ 0 & cot\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & -\frac{f}{f-n} & -\frac{nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

14. In (a), a sphere travels about a fixed camera in a circle. Suppose that the rotating sphere is rendered through the GPU pipeline and the camera periodically captures five images while the sphere is inside of the view frustum. If the captured images were overlapped, we would expect the result in (b). The distance between the camera and sphere is unchanged and therefore the sphere's size would remain the same wherever it is located on the circle. However, the overlapped images produced by the GPU rendering pipeline appear as in (c). As soon as the sphere enters the view frustum (the leftmost sphere), it looks the largest. If the sphere is on the $-n$ axis of the camera space (the sphere in the middle), it looks the smallest. Right before the sphere leaves the view frustum (the rightmost sphere), it looks the largest. Explain why.


(a)

(b)


(c)

Hint: Even though the Euclidean distances from the camera to the moving spheres are the same, their z -coordinates are different.

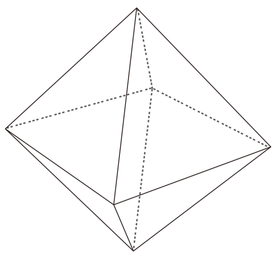
Chapter 6

1. Given an OpenGL ES program and its vertex shader shown below, fill in the boxes. Assume that `worldMat` includes a non-uniform scaling. The second argument of `glVertexAttribPointer` specifies the number of components in the attribute and the fifth argument specifies the stride.

```
#version 300 es

uniform mat4 worldMat, viewMat, projMat;

layout(location = 2) in vec3 position;
layout(location = 3) in vec3 normal;
layout(location = 7) in vec2 texCoord;

out vec3 v_normal;
out vec2 v_texCoord;

void main() {
     gl_Position = projMat * viewMat * worldMat * vec4(position, 1.0);
     v_normal = normalize(transpose(inverse(mat3(worldMat))) * normal);
     v_texCoord = texCoord;
}
```

```
struct Vertex
{
    glm::vec3 pos; // position
    glm::vec3 nor; // normal
    glm::vec2 tex; // texture coordinates
};
typedef GLushort Index;

glEnableVertexAttribArray(2); // position
glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE,
          sizeof(Vertex), (const GLvoid*) offsetof(Vertex, pos));

glEnableVertexAttribArray(3); // normal
glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE,
          sizeof(Vertex), (const GLvoid*) offsetof(Vertex, nor));

glEnableVertexAttribArray(7); // texture coordinates
glVertexAttribPointer(7, 2, GL_FLOAT, GL_FALSE,
          sizeof(Vertex), (const GLvoid*) offsetof(Vertex, tex));
```

2. The triangle mesh shown below is composed of eight triangles.



   (a) If we have the indexed representation of the mesh, we will invoke `glDrawElements(mode, count, type, indices)`. What will `mode` be? What will `count` be?

```
glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_SHORT, 0)
```

   (b) If we have the non-indexed representation of the mesh, we will invoke `glDrawArrays(mode, first, count)`. What will `mode` be? What will `count` be?

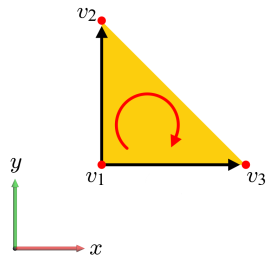
```
glDrawArrays(GL_TRIANGLES, 0, 24).
```

Chapter 7

1. Shown below is an object in NDC of the left-handed clip space.

   (a) For back-face culling, we consider only the x- and y-coordinates of each vertex. Assume that v_1 and v_2 have the same x-coordinate and v_1 and v_3 have the same y-coordinate. Is the triangle's winding order CW or CCW? Answer this question by drawing the 2D triangle.

   The vertices have the CW winding order.



   (b) The winding order is determined by checking the sign of a determinant. Is the determinant positive or negative?

   negative.

2. A viewport's corners are located at $(10, 20, 1)$ and $(100, 200, 2)$. The viewport transform is defined as a scaling followed by a translation.

   (a) Write the scaling matrix.

$$\begin{pmatrix} 45 & 0 & 0 & 0 \\ 0 & 90 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) Write the translation matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 55 \\ 0 & 1 & 0 & 110 \\ 0 & 0 & 1 & 1.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Our GL program invokes two functions: `glViewport(10, 20, 200, 100)` and `glDepthRangef(0, 1)`.

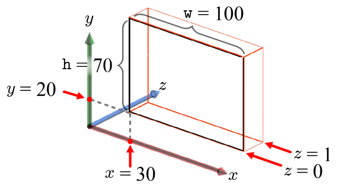
   (a) Write the scaling matrix for the viewport transform.

$$\begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

   (b) Write the translation matrix for the viewport transform.

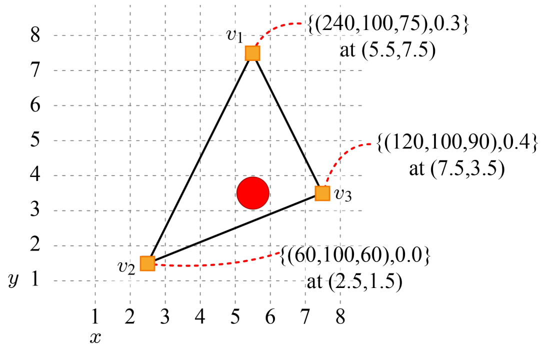
$$\begin{pmatrix} 1 & 0 & 0 & 110 \\ 0 & 1 & 0 & 70 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Shown below is a 3D viewport. Compute the viewport transform matrix.



$$\begin{pmatrix} 50 & 0 & 0 & 80 \\ 0 & 35 & 0 & 55 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

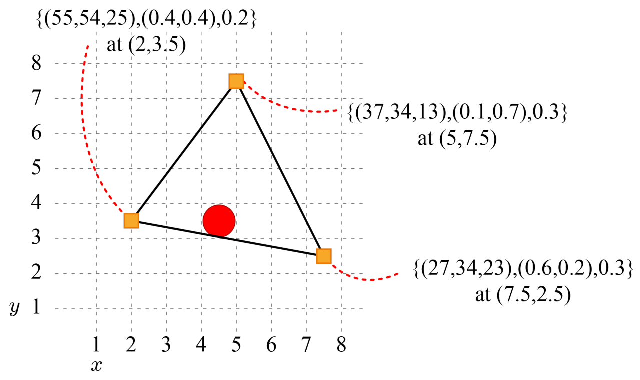
5. Shown below is a screen-space triangle. Each vertex is associated with $\{(R, G, B), z\}$. Compute R and z for the fragment at $(5.5, 3.5)$. Instead of taking the mechanical steps based on *slopes*, use the *linear interpolation* for the intersection of the left edge of the triangle with the scan line at y-coordinate 3.5. Along the scan line, also use the linear interpolation.



{(240,100,75),0.3} at (5.5,7.5)
{(120,100,90),0.4} at (7.5,3.5)
{(60,100,60),0.0} at (2.5,1.5)
v_1
v_2
v_3
x
y

R is 120, and z is 0.25.

6. Shown below is a screen-space triangle. The vertex attributes are the RGB color, 2D texture coordinates, and z. Interpolate the attributes for the fragment at $(4.5, 3.5)$. Instead of taking the mechanical steps based on *slopes*, use the *linear interpolation* for the intersection of the right edge of



{(55,54,25),(0.4,0.4),0.2} at (2,3.5)
{(37,34,13),(0.1,0.7),0.3} at (5,7.5)
{(27,34,23),(0.6,0.2),0.3} at (7.5,2.5)
x
y

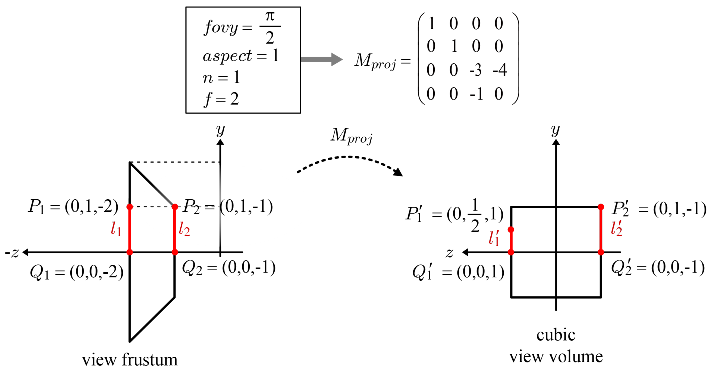
$\{(42, 44, 23), (0.45, 0.35), 0.25\}$.

7. Shown below is the projection matrix. Suppose that $n = 1$, $f = 2$, $fovy = \frac{\pi}{2}$, and the view frustum's width equals its height. Apply the projection transform to the camera-space points, $(0, 2, -2)$ and $(0, 0, -1)$. Then, apply the projection transform to $(0, 1, -1.5)$, which is the midpoint between $(0, 2, -2)$ and $(0, 0, -1)$. Is the result still the midpoint in NDC? Discuss why or why not.

$$\begin{pmatrix} \frac{cot\frac{fovy}{2}}{aspect} & 0 & 0 & 0 \\ 0 & cot\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

This is not the midpoint.

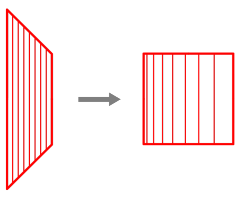
8. Shown below is an example of the projection matrix.



(a) Suppose that a camera-space point at $(0, 1, -1.5)$ is projection-transformed. Compute the transformed point.

$$\begin{pmatrix} 0 \\ \frac{2}{3} \\ \frac{1}{3} \\ 1 \end{pmatrix}$$

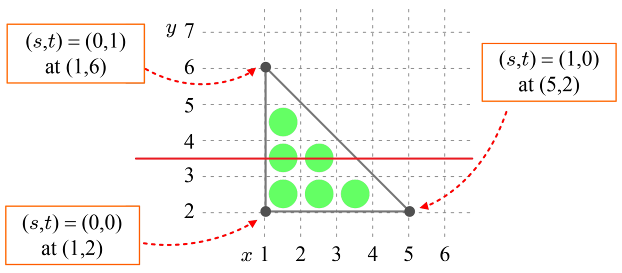
(b) With the result of (a), you may be able to guess the following figure. Discuss why this *non-linearity* happens.



Hint: Read [Note: Perspective correction].

Chapter 8

1. Consider the scan line at y-coordinate 3.5.



   (a) Compute the texture coordinates at the intersection points between the scan line and two edges.

   $(s,t) = (0, \frac{3}{8})$, $(\frac{5}{8}, \frac{3}{8})$.

   (b) Compute the texture coordinates of two fragments at the scan line.

   $(s,t) \ = \ (\frac{1}{8}\ , \frac{3}{8}),\ (\frac{3}{8}, \frac{3}{8})$.

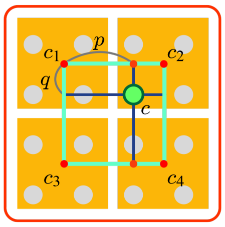
2. Suppose that the texture coordinate s is outside of the range $[0, 1]$. Assuming that the texture wrapping mode is *repeat*, write an equation that converts s into the range $[0, 1]$. Use the floor or ceiling function.

   $s' = s - \lfloor s \rfloor$

3. Suppose that a pixel is projected into (s', t') in the texture space. Using the floor or ceiling function, write the equations to compute the texel *index* for nearest point sampling. The index of a texel is the coordinates of its lower-left corner. For example, the index of the texel located at $(1.5, 3.5)$ is $(1, 3)$.

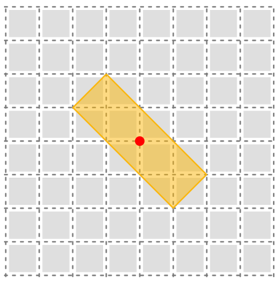
   $s'' = \lfloor s' \rfloor$
   $t'' = \lfloor t' \rfloor$

4. Shown below is a fragment projected into the texture space. It is surrounded by four texels. The color of each texel is denoted by c_i. Compute the fragment color, c, using bilinear interpolation.



$$c_{12} = (1-p)c_1 + pc_2$$
$$c_{34} = (1-p)c_3 + pc_4$$
$$c = (1-q)c_{12} + qc_{34}$$

5. A pixel is projected onto an 8×8-resolution image texture. On the left of the figure, the red dot represents the projected point, and the yellow rectangle is the footprint.


level 0

9	9	9	9	9	7	0	1
9	9	9	9	7	9	8	3
6	5	8	9	2	3	2	1
5	8	8	7	7	0	4	1
7	8	1	1	1	1	3	3
8	5	1	5	1	1	0	2
4	9	9	9	1	2	4	8
8	3	9	9	4	1	0	0

level 0

9	9	8	3
6	8	3	2
7	2	1	2
6	9	2	3

level 1

(a) The texture is a gray-scale image. The numbers in the level-0 and level-1 textures represent the texel intensities. Construct the level-2 and level-3 textures.

8	4
6	2

level 2

5

level 3

(b) Suppose that trilinear interpolation is used for texture filtering and λ is set to the length of the *longest* side of the footprint. Which levels are selected? Compute the filtered result at each level.

Levels 2 and 3 are selected. Level 2 is 5, and level 3 is 5.

(c) Suppose that trilinear interpolation is used for texture filtering and λ is set to the length of the *shortest* side of the footprint. Which levels are selected? Compute the filtered result at each level.

Levels 0 and 1 are selected. Level 0 is 4, and level 1 is 3.5.

6. Suppose that, for the OpenGL ES function `glTexParameteri(GLenum target, GLenum pname, GLint param)`, `target` is `GL_TEXTURE_2D` and `pname` is `GL_TEXTURE_MIN_FILTER`.

(a) If we choose `GL_LINEAR` for `param`, what problem might we have?

Aliasing artifact.

(b) Suppose that we choose `GL_LINEAR_MIPMAP_NEAREST` for `param`, which implies that we choose the nearest level in the mipmap. How many "linear interpolations" are performed in total? (Note that this question is not asking the count of bilinear interpolations.)

We do the linear interpolation three times.

(c) Suppose that we choose `GL_NEAREST_MIPMAP_LINEAR` for `param`. How many "linear interpolations" are performed in total?

We do the linear interpolation just once.

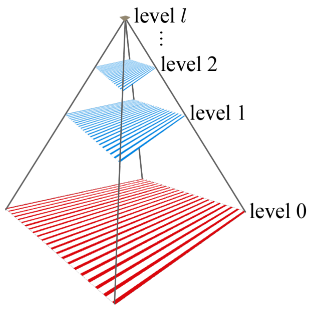
(d) Suppose that we choose `GL_LINEAR_MIPMAP_LINEAR` for `param`. How many "linear interpolations" are performed in total?

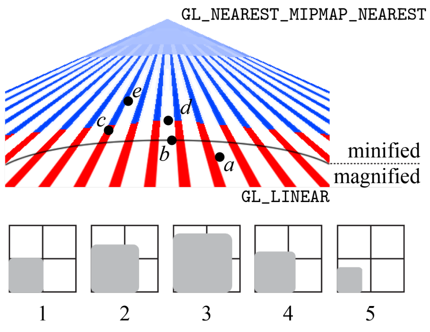
We do the linear interpolation seven times.

7. Some applications such as medical imaging require 3D texturing. Given a 3D image with a $2^l \times 2^l \times 2^l$ resolution, describe how to construct the mipmap. How many levels are in the mipmap? What is the size of the top-level image in the mipmap?

   $(l+1)$ levels are created and all the other answers are straight.

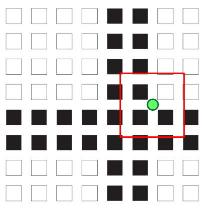
8. Shown below are a mipmap, a textured quad, and five cases of pixel projection, where the gray box represents a pixel's footprint and the 2×2 grid represents 4 texels. Make five alphabet-number pairs.





   $(a,5), (b,1), (c,4), (d,2), (e,3)$

9. Consider a checkerboard image composed of two gray-scale values, 0 (black) and 0.8 (light gray) in the range of $[0,1]$. The footprint of a projected pixel is square. Its center is exactly in the middle of 4 texels, and the side length is $2\sqrt{2}$.



(a) Suppose that each level in the mipmap is filtered by nearest point sampling and the filtered results are linearly interpolated. What is the final textured color? (If there are multiple texels whose distances to the pixel are the same, we choose the upper or upper-right one for nearest point sampling.)

0.6.

(b) Now suppose that each level is filtered by bilinear interpolation. What is the final textured color?

0.25.

Chapter 9

1. The Phong model shown below assumes a directional light source.

$$max(n \cdot l, 0)s_d \otimes m_d + (max(r \cdot v, 0))^{sh} s_s \otimes m_s + s_a \otimes m_a + m_e$$

   (a) How would you modify it for handling multiple directional light sources?

   Hint: Different lighting terms are handled differently.

   (b) How would you modify it for replacing the directional light source by a point light source? Assume that the light intensity a surface point receives is inversely proportional to the square of distance that the light has traveled.

   Hint: The light vector needs to be computed for each vertex or fragment.

2. At line 16 of Sample code 9-1, `v_view` is a unit vector, but line 19 of Sample code 9-2 normalizes it again. Explain why.

   Hint: Consider the rasterization stage.

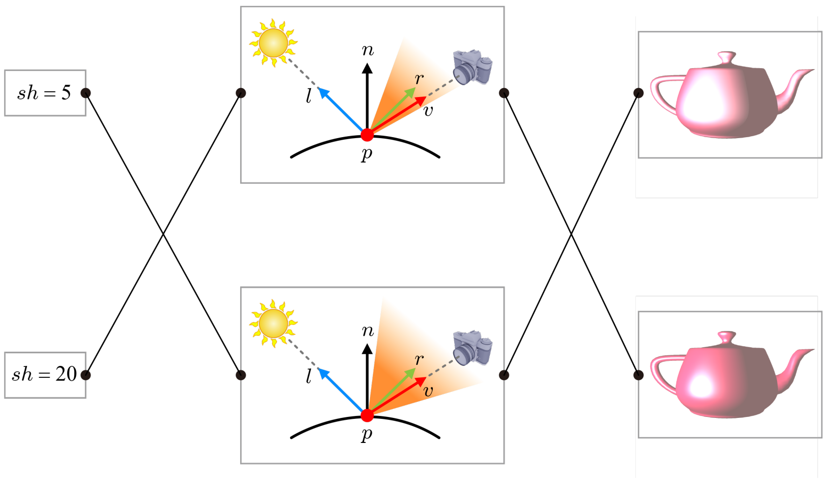
3. At line 24 of Sample code 9-2, why do we need `max`?

   Hint: Read the textbook.

4. For specular reflection, it is necessary to compute the refection vector r using the surface normal n and the light vector l. Write the equation for r using the dot product of n and l.

   $r = 2n(n \cdot l) - l$

5. Consider the figure shown below. It is for the specular reflection term of the Phong lighting model: $(max(r \cdot v, 0))^{sh} s_s \otimes m_s$. Illustrated in the middle column are the cones representing the ranges where we can see the highlights. Connect the black dots. For example, if you think that a smaller sh leads to a smaller cone, connect the upper-left box to the upper-middle box.



Chapter 10

1. Consider four triangles competing for a pixel location. They have distinct depth values at the pixel location. If the triangles are processed in an arbitrary order, how many times would the z-buffer be updated on average for the pixel location?

   $\simeq 2.08$.

2. You have three surface points competing for a pixel location. Their RGBA colors and z-coordinates are given as follows: $\{(1, 0, 0, 0.5), 0.25\}$, $\{(0, 1, 0, 0.5), 0.5\}$, and $\{(0, 0, 1, 1), 0.75\}$. They are processed in the back-to-front order. Compute the final color of the pixel.

   (0.5,0.25,0.25).

3. Consider three triangles in the viewport. They are all perpendicular to the z-axis. The red triangle is behind the green, which is behind the blue. Three fragments with RGBA colors, $(1, 0, 0, 1)$, $(0, 1, 0, 0.5)$ and $(0, 0, 1, 0.5)$, compete for a pixel location and they are processed in the back-to-front order. Compute the final color of the pixel.

   (0.25,0.25,0.5).

4. Consider five fragments competing for a pixel location. Their RGBA colors and z-coordinates are given as follows:
   $f_1 = \{(1, 0, 0, 0.5), 0.2\}$
   $f_2 = \{(0, 1, 1, 0.5), 0.4\}$
   $f_3 = \{(0, 0, 1, 1), 0.6\}$
   $f_4 = \{(1, 0, 1, 0.5), 0.8\}$
   $f_5 = \{(0, 1, 0, 1), 1.0\}$

   (a) What is the correct order of processing the fragments?

   The correct order is either $f_3 \rightarrow f_5 \rightarrow f_4 \rightarrow f_2 \rightarrow f_1$ or $f_5 \rightarrow f_3 \rightarrow f_4 \rightarrow f_2 \rightarrow f_1$.

   (b) Compute the final color of the pixel.

   $(0.5, 0.25, 0.5)$.

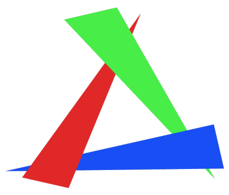
5. Fog is often used to enhance the realism of outdoor scenes. The simplest implementation is a *linear* fog, which starts from the near plane and ends at the far plane of the view frustum. The objects located at the near plane are clearly visible whereas objects at the far plane are completely obscured by the fog. Such a linear fog can be described by the following blending equation:

$$c = fc_f + (1 - f)c_o$$

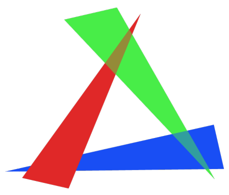
where c is the fogged color, f represents the fog factor which increases with the distance from the viewer, c_f is the fog color, and c_o is the fragment color. Define the fog factor f as a function of the near plane's depth (denoted as N) and the far plane's depth (denoted as F).

If the distance of an object surface from the camera is d, f is defined to be $\frac{d-N}{F-N}$.

6. Consider three polygons shown below. They occlude one another in a circle.



(a) Assume that the polygons are translucent and the rendering order is red, green, and then blue. Sketch the rendered result.

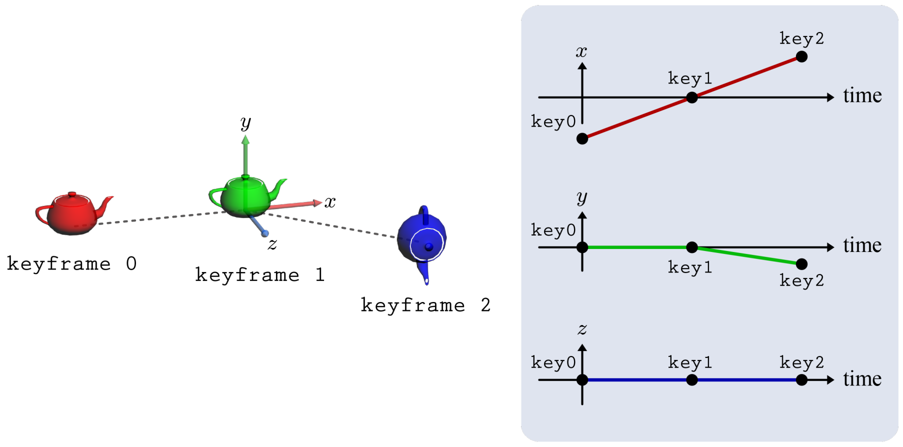


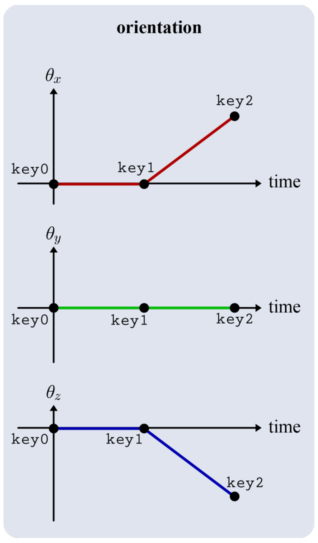
(b) What problem do you find? How would you resolve the problem?

Hint: With the triangles untouched, there is no solution.

Chapter 11

1. Shown below are teapots at three keyframes and the position graphs for keyframe animation. Draw the orientation graphs.





2. Let $\{u, v, n\}$ represent the object-space basis. Suppose that an object is rotated about the n-axis by θ_n and then rotated about the u-axis by θ_u. Describe the object-space Euler transform, $R_u(\theta_u)R_n(\theta_n)$, as a combination of world-space rotation matrices. (The rotation matrices about the world-space x-, y-, and z-axes are denoted as R_x, R_y, and R_z, respectively.)

$$R_u(\theta_u)R_n(\theta_n) = R_z(\theta_n)R_x(\theta_u)$$

3. Consider rotating $(0, 1, 0)$ about $(1, 0, 1)$ by $90°$.

   (a) Write the quaternion that represents "rotation about $(1, 0, 1)$ by $90°$." [Hint: The imaginary part of a quaternion contains the sine function and the real part contains the cosine function.]

   $\mathbf{q} = (1/2, 0, 1/2, 1/2)$.

   (b) In order to rotate $(0, 1, 0)$ using the above quaternion, $(0, 1, 0)$ should also be represented in a quaternion. Write it.

   $\mathbf{p} = (0, 1, 0, 0)$

   (c) When $\mathbf{q}$ and $\mathbf{p}$ denote the quaternions in (a) and (b), respectively, $\mathbf{qpq}^*$ defines the rotation of $(0, 1, 0)$ about $(1, 0, 1)$ by $90°$. Compute $\mathbf{qp}$ and $\mathbf{q}^*$. [Hint: $ij = k$.]

$$\begin{aligned}\mathbf{qp} &= (-\tfrac{1}{2}, \tfrac{1}{\sqrt{2}}, \tfrac{1}{2}, 0)\\ \mathbf{q}^* &= (-\tfrac{1}{2}, 0, -\tfrac{1}{2}, \tfrac{1}{\sqrt{2}})\end{aligned}$$

4. Let $\mathbf{q}$ denote the quaternion for "rotation about a unit vector $\mathbf{u}$ by θ." Prove that $-\mathbf{q}$ represents the same rotation.

Chapter 12

1. In Fig. 12.3, the screen-space ray starts from $(x_s, y_s, 0)$ and the camera-space ray starts from $(x_c, y_c, -n)$.

   (a) Using the *inverse* of the viewport transform, compute the clip-space ray's start point in NDC.

   $(\frac{2x_s}{\mathtt{w}} - 1, \frac{2y_s}{\mathtt{h}} - 1, -1)$.

   (b) Using the fact that the answer in (a) is identical to the point in Equation (12.1), compute x_c and y_c.

   x_c is $\frac{n}{m_{11}}(\frac{2x_s}{\mathtt{w}} - 1)$, and y_c is $\frac{n}{m_{22}}(\frac{2y_s}{\mathtt{h}} - 1)$

2. Suppose that, for object picking, the user clicks exactly the center of the viewport.

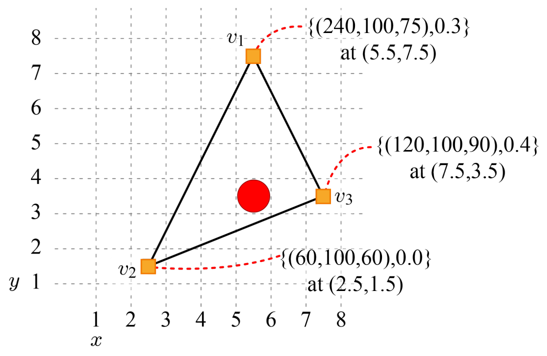
   (a) The view-frustum parameters are given as follows: $n = 12$, $f = 18$, $fovy = 120°$, and $aspect = 1$. Represent the camera-space ray in a parametric equation of t. [Hint: No transform between the screen and camera spaces is needed. The camera-space ray can be intuitively defined.]

   $(0, 0, -12 - t)$.

   (b) Imagine a camera-space bounding sphere. Its radius is 2 and center is at $(0, -1, -14)$. Compute the parameter t at the first intersection between the ray and the bounding sphere, and also compute the 3D coordinates of the intersection point.

   $$t = 2 - \sqrt{3} \qquad (0, 0, -14 + \sqrt{3})$$

3. Shown below is a screen-space triangle, each vertex of which is associated with $\{(R, G, B), z\}$.



(a) Compute the barycentric coordinates of the red dot at $(5.5, 3.5)$ in terms of v_1, v_2, and v_3.

$(\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$

(b) Using the barycentric coordinates, compute R and z at $(5.5, 3.5)$.

0.25

4. The last step in Fig. 12.15 is the *inverse* of the world transform. Suppose that the original world matrix is a rotation (denoted as R_0) followed by a translation (denoted as T) and $\{p_1, p_2, p_3, \ldots, p_n\}$ represents the finger's trajectory.

(a) Let R_1 denote the rotation matrix computed using p_1 and p_2. Define the inverse world transform used to compute R_1.

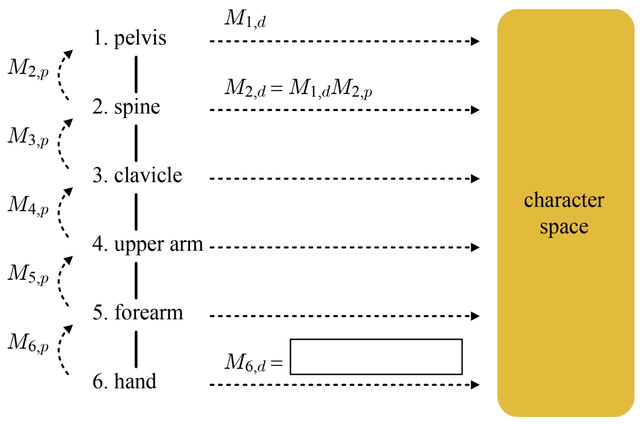
It is $(TR_0)^{-1}$

(b) Let R_2 denote the rotation matrix computed using p_2 and p_3. Define the inverse world transform used to compute R_2.

$(TR_0R_1)^{-1}$,

Chapter 13

1. Shown below is the bone hierarchy augmented with the transforms for the default pose.



   (a) Describe what $M_{2,p}$ does.

   It transforms the vertex defined in the bone space of spine into the bone space of its parent, pelvis.

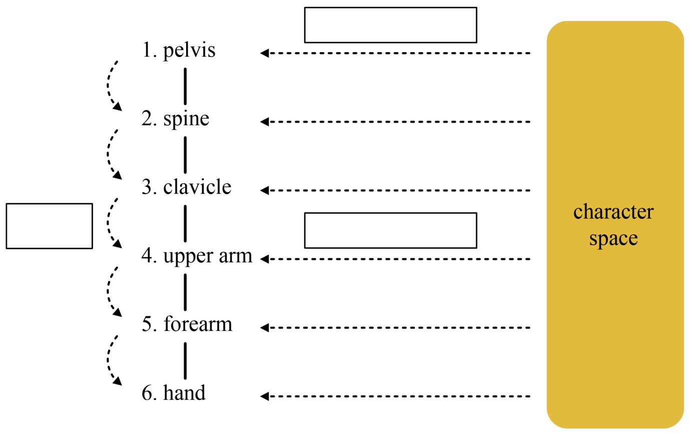
   (b) Describe what $M_{2,d}$ does.

   It transforms the vertex defined in the bone space of spine into the character space.

   (c) Fill in the blank for $M_{6,d}$.

   $M_{6,d} = M_{5,d}M_{6,p}$

(d) Shown below is the inverse process. Fill in the boxes.

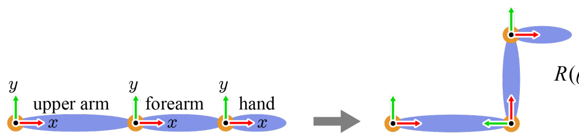


$M_{4,p}^{-1}$

$M_{1,d}^{-1}$

$M_{4,d}^{-1} = M_{4,p}^{-1} M_{3,d}^{-1}$

2. Shown on the left is the default pose. Let us take the bone space of the upper arm as the character space. The bone-space origins of the forearm and hand are (12,0) and (22,0), respectively, with respect to the character space. From the default pose, the forearm is rotated by 90°, and the hand is rotated by −90°, to define the animated pose.



$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(a) In the default pose, compute the to-parent matrices of the forearm and hand ($M_{f,p}$ and $M_{h,p}$).

$$M_{f,p} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{h,p} = \begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(b) Using $M_{f,p}$ and $M_{h,p}$, compute the matrices, $M_{f,d}$ and $M_{h,d}$, which respectively transform the vertices of the forearm and hand into the character space.

$$M_{f,d} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{h,d} = \begin{pmatrix} 1 & 0 & 22 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(c) Compute $M_{f,d}^{-1}$ and $M_{h,d}^{-1}$.

$$M_{f,d}^{-1} = \begin{pmatrix} 1 & 0 & -12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$M_{h,d}^{-1} = \begin{pmatrix} 1 & 0 & -22 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(d) Compute the local transform matrices of the forearm and hand ($M_{f,l}$ and $M_{h,l}$).

$$M_{f,l} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$M_{h,l} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(e) Compute the matrix, $M_{f,a}$, which animates the vertices of the forearm and transforms them back to the character space.

$$M_{f,a} = \begin{pmatrix} 0 & -1 & 12 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

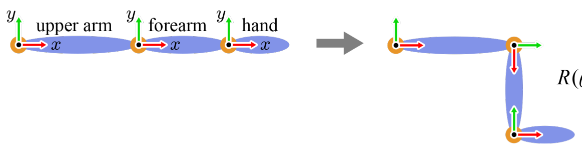
(f) Compute the matrix, $M_{h,a}$, which animates the vertices of the hand and transforms them back to the character space.

$$M_{h,a} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{pmatrix}$$

(g) Consider a vertex v whose coordinates in the forearm's bone space are (8,0). It is affected by two bones, the forearm and hand, which have the same blend weights. Using the skinning algorithm, compute the character-space position of v in the animated pose.

$(11, 9)$.

3. Consider the same arm as in Problem 2. The forearm is rotated by $-90°$, and the hand is rotated by $90°$, to define the animated pose. Answer the same questions as (a) through (g) of Problem 2.



$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(a) In the default pose, compute the to-parent matrices of the forearm and hand ($M_{f,p}$ and $M_{h,p}$).

$$M_{f,p} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{h,p} = \begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(b) Using $M_{f,p}$ and $M_{h,p}$, compute the matrices, $M_{f,d}$ and $M_{h,d}$, which respectively transform the vertices of the forearm and hand into the character space.

$$M_{f,d} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{h,d} = \begin{pmatrix} 1 & 0 & 22 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(c) Compute $M_{f,d}^{-1}$ and $M_{h,d}^{-1}$.

$$M_{f,d}^{-1} = \begin{pmatrix} 1 & 0 & -12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{h,d}^{-1} = \begin{pmatrix} 1 & 0 & -22 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(d) Compute the local transform matrices of the forearm and hand ($M_{f,l}$ and $M_{h,l}$).

$$M_{f,l} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{h,l} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(e) Compute the matrix, $M_{f,a}$, which animates the vertices of the forearm and transforms them back to the character space.

$$M_{f,a} = \begin{pmatrix} 0 & 1 & 12 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

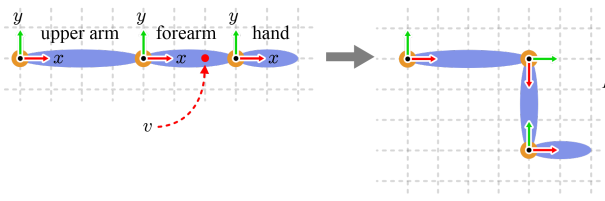
(f) Compute the matrix, $M_{h,a}$, which animates the vertices of the hand and transforms them back to the character space.

$$M_{h,a} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{pmatrix}$$

(g) Consider a vertex v whose coordinates in the forearm's bone space are (8,0). It is affected by two bones, the forearm and hand, which have the same blend weights. Using the skinning algorithm, compute the character-space position of v in the animated pose.

$(11, -9)$.

4. From the default pose shown on the left, the forearm is rotated by $-90°$, and the hand is rotated by $90°$, to define the animated pose. Suppose that v is affected by both forearm and hand.



$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(a) What are v's coordinates in the bone space of the forearm?

$v_f = (2, 0)$

(b) Using two matrices, show the process of computing the coordinates of "v rotated by the forearm" in the bone space of the upper arm.

$$v'_f = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

(c) What are v's coordinates in the bone space of the hand?

$v_h = (-1, 0)$

(d) Using four matrices, show the process of computing the coordinates of "v rotated by the hand" in the bone space of the upper arm.

$$v''_h = \begin{pmatrix} 3 \\ -3 \\ 1 \end{pmatrix}$$

(e) The forearm and hand have the blend weights of 80% and 20%, respectively, on v. Compute the coordinates of v in the bone space of the upper arm.

(3.8, −2.2)

5. Shown below is the pseudocode for combining skinning and keyframe animation for a human character.

```
 1: for each bone // default pose
 2:     compute Md-
 3:
 4: for each frame
 5:     for each bone i // animated pose
 6:         interpolate key data
 7:         compute Ma
 8:         combine Ma with Md- to define Mi
 9:         store Mi in the matrix palette
10:     invoke vertex shader for skinning
```

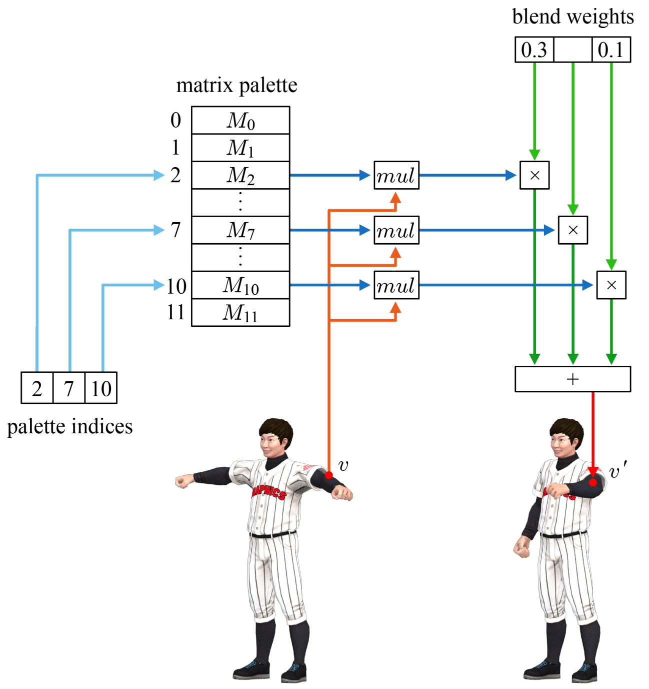
(a) At line 6, what kinds of key data are interpolated?

If $M_{i,a} = M_{i-1,a}M_{i,p}M_{i,l}$, let $M_{i,p,l}$ denote the combination of $M_{i,p}$ and $M_{i,l}$. Its upper-left 3×3 sub-matrix represents a 'combined' rotation, and the fourth column represents a 'combined' translation. For a keyframe, the rotation component of $M_{i,p,l}$ is stored as a quaternion, and the translation component is stored as a vector. They form the key data. For each in-between frame of animation, the skeleton hierarchy is traversed in a top-down fashion to compute $M_{i,a}$ for each bone. The quaternions and translational vectors stored in the keyframes are independently interpolated. The interpolated quaternion is converted into a matrix, and the interpolated translation vector fills the fourth column of the matrix. This matrix is combined with $M_{i-1,a}$ so as to complete $M_{i,a}$.

(b) The vertex shader invoked at line 10 will perform texturing and lighting in addition to skinning. List all data stored in the vertex array.

The object-space position is always required. For lighting and texturing, normal and texture coordinates are needed. For skinning, the palette indices and the blend weights are needed.

6. The figure shown below depicts how skinning is implemented.



default pose

animated pose

(a) How many bones does the character have?

12

(b) How many bones affect a vertex?

3

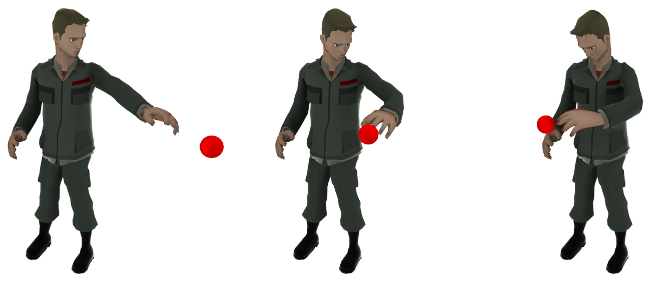
(c) Fill in the blank at the upper-right corner.

0.6

(d) In the matrix palette, M_i is a combination of two matrices: one remains fixed throughout the entire animation whereas the other is updated for every frame. What are the matrices? What do they do?

$M_i = M_{i,a}M_{i,d}^{-1}$. Given the default pose, $M_{i,d}^{-1}$ of each bone is computed once and remains fixed. For each frame of animation, the skeleton hierarchy is traversed in a top-down fashion to compute $M_{i,a}$ for each bone. A character-space vertex in the default pose is transformed to the bone space by $M_{i,d}^{-1}$. Then, it is animated and transformed back to the character space by $M_{i,a}$:

7. In the figure shown below, dynamic gazing is implemented. Involved in dynamic gazing is the 3-DOF joint connecting the head and neck. Describe an analytic solution for dynamic gazing.



Suppose the bone space of the head, the origin of which is located at the head-neck joint. Also suppose that the head bone is along the x-axis, and the character is facing in the negative z direction. (The y-axis is perpendicular to both of them.) Now consider a flying object. Its world-space position is transformed to the bone space of the head. The vector connecting the eye and the object is computed and normalized. Let us call the vector v. Then, as presented in Fig. 11.20-(c), the rotation axis is obtained using the cross product of -z and v, and the rotation angle is obtained using the dot product of -z and v. Then, the head is rotated using the rotation axis and angle. (For realistic animations, the rotation angle needs to be constrained within a range such that the character cannot gaze at the object outside the field of view.)

Chapter 14

1. Vertex array data.

   (a) In order to apply tangent-space normal mapping to a character, what data does the vertex array store?

   The object-space position is always required. For tangent-space normal mapping, normal, texture coordinates, and tangent are needed. Bitangent is optional. It can be computed by the vertex shader.

   (b) In addition, the character is going to be skin-animated. What data does the vertex array store?

   For skinning, the palette indices and the blend weights need to be added to the above.

2. Shown below is the vertex shader for tangent-space normal mapping. Fill in the boxes.

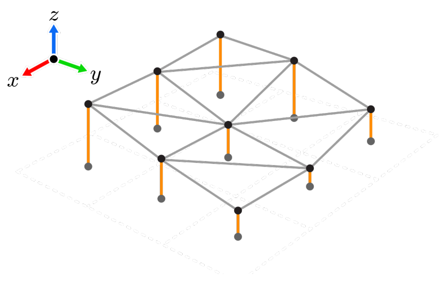
```
#version 300 es

uniform mat4 worldMat, viewMat, projMat;
uniform vec3 eyePos, lightDir;

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texCoord;
layout(location = 3) in vec3 tangent;

out vec3 v_lightTS, v_viewTS;
out vec2 v_texCoord;

void main() {
    vec3 worldPos = (worldMat * vec4(position, 1.0)).xyz;
    vec3 Nor = normalize(transpose(inverse(mat3(worldMat))) * normal);
    vec3 Tan =                                                              ;
    vec3                                   ;
    mat3 tbnMat = transpose(mat3(Tan, Bin, Nor)); // row major

    v_lightTS = tbnMat * normalize(lightDir);
    v_viewTS =                                                   ;

    v_texCoord = texCoord;
    gl_Position = projMat * viewMat * vec4(worldPos, 1.0);
}
```

See Sample code 14-3.

3. Suppose that the per-vertex tangent-space basis, $\{T, B, N\}$, is stored in the vertex array and passed to the vertex shader. Write the matrix that converts a tangent-space normal vector into the world space.

$$M = \begin{pmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{pmatrix}$$

4. In Fig. 14.6, we use the finite difference method to compute the normal at a height-field point. Describe another method that computes the normal using the surface normals of the triangles sharing the point.



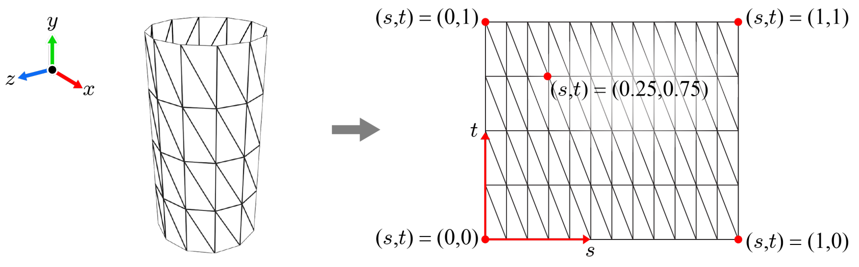
Suppose that, as shown above, a height-field point is shared by six triangles. We can compute the normals for all triangles, and then assign their mean to the point. In other triangulation methods, a height-field point can be shared by different numbers of triangles, but we can use the same method of averaging the normals.

5. Shown below is the fragment shader for tangent-space normal mapping. Fill in the boxes.

```
#version 300 es

precision mediump float;

uniform sampler2D colorMap, normalMap;
uniform vec3 srcDiff; // Sd

in vec3 v_lightTS;
in vec2 v_texCoord;

layout(location = 0) out vec4 fragColor;

void main() {
    // normal map access
    vec3 normal = ________________________________________;

    vec3 light = _____________________;

    // diffuse term
    vec3 matDiff = ________________________________________;
    vec3 diff = max(dot(normal, light), 0.0) * srcDiff * matDiff;

    fragColor = vec4(diff, 1.0);
}
```

See lines 16, 18, and 21 of Sample code 14-4.

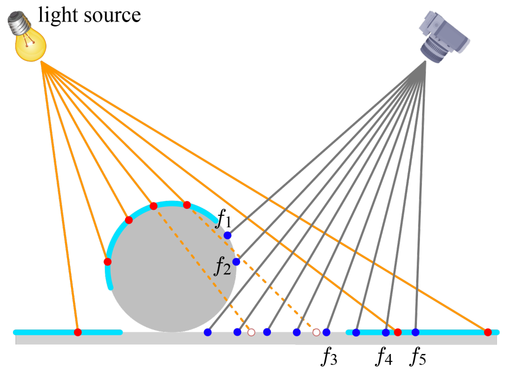
6. Consider a cylinder and its parameterization shown below. Suppose that the cylinder axis equals the y-axis of the coordinate system. Normal mapping is to be applied to its surface. Describe how you can compute a tangent space for each vertex (x, y, z). (Do not use any information from other vertices but use only (x, y, z) of the vertex.)

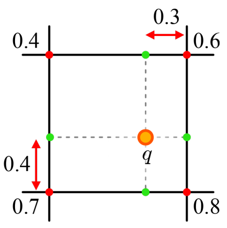


As the axis of the cylinder is the y-axis, the vertex normal will be $(x, 0, z)$. (It is not yet normalized.) It corresponds to N. We need to compute T and B. We can safely make every vertex have the same B, $(0, 1, 0)$. (Because the axis of the cylinder is the y-axis, such B is tangent to the cylindrical surface everywhere.) Then, T is made to be parallel to the zx-plane because it is perpendicular to B. As T is perpendicular to N, it can be simply set to $(z, 0, -x)$. Observe that T, B, and N are mutually orthogonal. As a final step, B and N need to be normalized. In conclusion, $T = \frac{(z,0,-x)}{\sqrt{x^2+z^2}}$, $B = (0, 1, 0)$, and $N = \frac{(x,0,z)}{\sqrt{x^2+z^2}}$.

Chapter 15

1. Shadow map filtering.





(a) In the figure shown on the left, the red and blue dots represent the points sampled in the first and second passes, respectively. Assume that nearest point sampling is used for shadow map filtering and biasing is not adopted. For each of five fragments, f_1 through f_5, determine if it will be shadowed or fully lit.

Whereas f_1, f_2, f_3, and f_5 are shadowed, f_4 is fully lit.

(b) In the figure shown on the right, q is a fragment projected into the shadow map. Its depth is 0.5. The values attached to the texels denote the depths stored in the shadow map. What is the fragment's visibility returned by the PCF algorithm?

0.88.

2. Shown below is the first-pass vertex shader for shadow mapping, where `lightViewMat` and `lightProjMat` respectively represent the view and projection matrices with respect to the light source. Fill in the box.

```
#version 300 es

uniform mat4 worldMat;
uniform mat4 lightViewMat, lightProjMat;

layout(location = 0) in vec3 position;

void main() {
    gl_Position = [                                        ];
}
```

See Sample code 15-1.

3. A problem that could be encountered in shadow mapping is called *Peter Panning*. Suppose that a sphere is placed on a planar surface. In the rendered image, the shadow generated in the planar surface may appear to be disconnected from the sphere that casts the shadow to the surface. The sphere looks as if it were floating above the shadow. When would you encounter this problem?

A large bias often leads to incorrect shadows. Some areas that should be shadowed are erroneously taken as lit, and consequently the shadow appears smaller than desired. If the bias becomes overlay large, the occluder can be separated from the shadow.

4. Shown below is the second-pass vertex shader for shadow mapping. It converts the clip-space homogeneous coordinates (x, y, z, w) into $(0.5x + 0.5w, 0.5y + 0.5w, 0.5z + 0.5w, w)$ and stores them into `v_shadowCoord`. Fill in the boxes.

```
#version 300 es

uniform mat4 worldMat, viewMat, projMat;
uniform mat4 lightViewMat, lightProjMat;
uniform vec3 lightPos;

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texCoord;

out vec3 v_normal, v_light;
out vec2 v_texCoord;
out vec4 v_shadowCoord;

const mat4 tMat = mat4(
        0.5, 0.0, 0.0, 0.0,



);

void main() {
    v_normal = normalize(transpose(inverse(mat3(worldMat))) * normal);
    vec3 worldPos = (worldMat * vec4(position, 1.0)).xyz;
    v_light = normalize(lightPos - worldPos);
    v_texCoord = texCoord;

    // for shadow map access and depth comparison
    v_shadowCoord =                                                    ;
    gl_Position =                                                      ;
}
```

See Sample code 15-4.

5. Shown below is the second-pass fragment shader for 'biased' shadow mapping. Fill in the boxes.

```
#version 300 es

precision mediump float;
precision mediump sampler2DShadow;

uniform sampler2D colorMap;
uniform sampler2DShadow shadowMap;
uniform vec3 srcDiff;

in vec3 v_normal, v_light;
in vec2 v_texCoord;
in vec4 v_shadowCoord;

layout(location = 0) out vec4 fragColor;

const float offset = 0.005;

void main() {
    vec3 normal = normalize(v_normal);
    vec3 light = normalize(v_light);

    // diffuse term
    vec3 matDiff = texture(colorMap, v_texCoord).rgb;
    vec3 diff = max(dot(normal, light), 0.0) * srcDiff * matDiff;

    vec4 offsetVec = [                                        ];
    float visibility = textureProj(shadowMap, [                   ]);

    fragColor = vec4(visibility * diff, 1.0);
}
```

See Sample code 15-6

Chapter 16

1. In the specular term of the Phong model, the reflection vector is defined to be $2n(n \cdot l) - l$, where n is the surface normal and l is the light vector. In ray tracing, the reflection ray's direction is defined to be $I - 2n(n \cdot I)$, where I is the primary ray's direction. What causes this difference?

   I is incident on the surface, but l leaves the surface.

2. Consider applying ray tracing to a sphere of radius 2, which is centered at the origin of the coordinate system.

   (a) A ray is fired from $(10, 1, 0)$ with the direction vector $(-1, 0, 0)$. Represent the ray in a parametric equation of t.

   $(10 - t, 1, 0)$.

   (b) Using the implicit equation of the sphere, $x^2 + y^2 + z^2 - 2^2 = 0$, and the parametric equation of the ray, compute the intersection point between the sphere and the ray.

   $$t = 10 - \sqrt{3} \qquad (\sqrt{3}, 1, 0)$$

   (c) In order to compute the reflection ray, the surface normal, n, at the intersection is needed. How would you compute n in this specific example?

   For a sphere centered at the origin of the coordinate system, the surface normal at a point is obtained by taking the point and normalizing it. Therefore, the surface normal is $(\frac{\sqrt{3}}{2}, \frac{1}{2}, 0)$.

   (d) The reflection ray's direction is defined to be $I - 2n(n \cdot I)$, where I represents the primary ray's. Compute $I - 2n(n \cdot I)$.

   $$r \;=\; I - 2n(n \cdot I) = \;(\tfrac{1}{2}, \tfrac{\sqrt{3}}{2}, 0)$$

3. Light mapping is usually combined with image texturing at run time to determine the diffuse reflection.

   (a) The light map can store the diffuse reflection, instead of the irradiance, so as to avoid the run-time combination of irradiance and diffuse reflectance. This would lead to an improved run-time performance but has a disadvantage. What is it?

   The brick-wall example shown in this chapter would give the most easy-to-understand explanation why the image texture and the light map are separated. When rendering a wide brick wall, a small-size brick-wall image texture is tiled through repeat or mirrored-repeat modes. On the other hand, the light map usually has a low resolution because diffuse reflections change slowly across the surface. Therefore, using two small textures (one is the image texture and the other is the light map) is more space-efficient than using a larger combined texture.

   (b) Light mapping is often called *dark mapping* because the pixel lit by the light map is darker than the unlit texel of the image texture. Why does this happen? How would you resolve this problem?

   It happens because the maximum irradiance value stored in the light map is 1. Unless the irradiance value is 1, the image texture is darkened. To increase the brightness, the combined color often needs to be multiplied by some factor greater than one. If the multiplied value becomes greater than one, it can be clamped to one.

4. Consider capturing six images for a cube map. Each image is generated using the view and projection transforms.

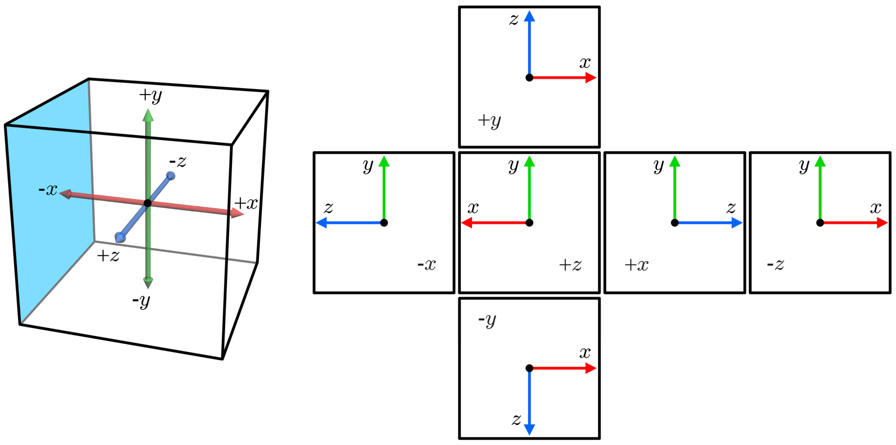
   (a) Is each image generated using a different view transform? Discuss why it is or is not.

   The view transform is defined by **EYE**, **AT**, and **UP**. Even though **EYE** is shared by six faces of the cube map, each face is associated with its own **AT** and **UP**, and therefore each face requires a distinct view transform.

   (b) Is each image generated using a different projection transform? Discuss why it is or is not.

   The environment is captured with $fovy$ set to 90° and *aspect* set to 1. It is because each face of the cube map covers a 90° field of view both vertically and horizontally. It is also natural to use the same n and f for the six faces. Therefore the projection transform is identical for the six faces.

5. Shown below is the unfolded cube map of six faces. Suppose that a reflection ray is computed and its direction is along $(0.5, 0.4, -0.2)$.



(a) Which face of the cube map does the reflection ray hit?

face $+x$ is selected.

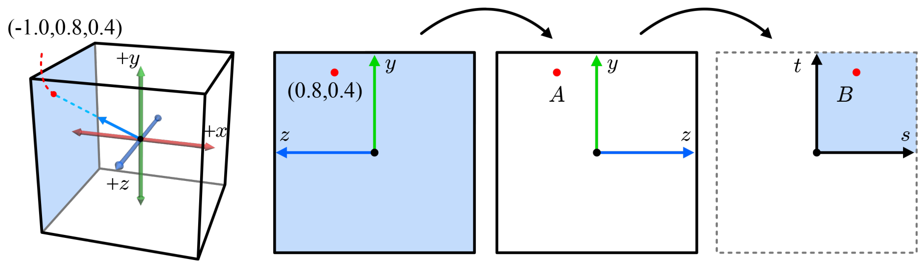
(b) What are the 2D coordinates of the intersection point between the reflection ray and the face?

The point hit by R has the yz-coordinates $(0.8, -0.4)$ at face $+x$.

(c) Compute the texture coordinates corresponding to the intersection point.

(0.3, 0.9)

6. Shown below is the process of computing the texture coordinates, (s, t), for cube mapping, which is done automatically by the GPU. Suppose that the reflection ray hits face -x at $(-1.0, 0.8, 0.4)$. Write the coordinates of A and B.



$A = (0.8, -0.4)$ and $B = (0.9, 0.3)$.

Chapter 17

1. The *cubic* Bézier curve has the equation, $(1-t)^3p_0+3t(1-t)^2p_1+3t^2(1-t)p_2+t^3p_3$. Write the equation of the *quartic* (degree-4) Bézier curve defined by $\{p_0,p_1,p_2,p_3,p_4\}$.

   It is $(1-t)^4p_0+4t(1-t)^3p_1+6t^2(1-t)^2p_2+4t^3(1-t)p_3+t^4p_4$.

2. Consider a quintic (degree-5) Bézier curve. How many control points are needed? For each control point, write the Bernstein polynomial over t in $[0,1]$.

   We need six control points, $\{p_0,p_1,p_2,p_3,p_4,p_5\}$, and the equation is defined as follows:

   $$p(t)=(1-t)^5p_0+5t(1-t)^4p_1+10t^2(1-t)^3p_2+10t^3(1-t)^2p_3+5t^4(1-t)^1p_4+t^5p_5$$

3. You are given three 2D points $\{(1,0),(0,1),(-1,0)\}$.

   (a) Assuming that the point at $(0,1)$ is associated with parameter 0.5, compute the control points of the quadratic Bézier curve that passes through all three points.

   $(0,2)$.

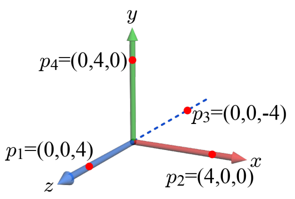
   (b) On the Bézier curve, compute the coordinates of the point whose parameter is 0.75.

   $(\frac{1}{2},\frac{3}{4})$

4. Consider a spline composed of two cubic Bézier curves defined by the control point sets, $\{p_0,p_1,p_2,p_3\}$ and $\{q_0,q_1,q_2,q_3\}$, where $p_3=q_0$. If they have the same tangent vector at their junction, the spline is called continuous. What is the necessary condition that makes the spline continuous? Describe the condition as an equation of p_2, p_3, q_0, and q_1.

   $p_3-p_2=q_1-q_0$

5. In the figure shown below, the camera is moving along the quadratic Bézier curve $p(t)$ defined by the control points, p_1, p_2, and p_3, whereas **AT** is moving along the linear path $q(t)$ connecting the origin and p_4. **UP** is fixed to the y-axis of the world space.



(a) Both $p(t)$ and $q(t)$ are defined in parameter t in the range $[0, 1]$. Compute the points on $p(t)$ and $q(t)$ when $t = 0.5$.

$p(t) = (2, 0, 0)$. $q(t) = (0, 2, 0)$

(b) Compute the basis of the camera space when $t = 0.5$.

n	$(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0)$
u	$(0, 0, -1)$
v	$(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$

(c) Compute the 4×4 translation and rotation matrices defining the view matrix when $t = 0.5$.

The translation is defined by **EYE**:

$$\begin{pmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The rotation is defined by u, v, and n:

$$\begin{pmatrix} 0 & 0 & -1 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6. A Bézier patch is defined by the control point matrix shown below. In its domain, the u- and v-axes run horizontally and vertically, respectively.

$$\begin{pmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{pmatrix} = \begin{pmatrix} (0,0,6) & (0,3,3) & (0,6,6) \\ (3,0,0) & (3,3,0) & (3,6,0) \\ (6,0,0) & (6,3,0) & (6,6,0) \end{pmatrix}$$

(a) Compute the 3D point when $(u, v) = (1, 0)$.

$(0, 6, 6)$.

(b) Using the method of "repeated bilinear interpolations," compute the 3D point when $(u, v) = (0.5, 0.5)$.

$(3, 3, 9/8)$.

7. Consider a Bézier patch, whose degrees in terms of u and v are three and two, respectively. The control point matrix is given as follows:

$$\begin{pmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \\ p_{30} & p_{31} & p_{32} \end{pmatrix} = \begin{pmatrix} (0,0,4) & (0,3,4) & (0,6,4) \\ (3,0,0) & (3,3,0) & (3,6,0) \\ (6,0,0) & (6,3,0) & (6,6,0) \\ (5,0,4) & (5,3,4) & (5,6,4) \end{pmatrix}$$

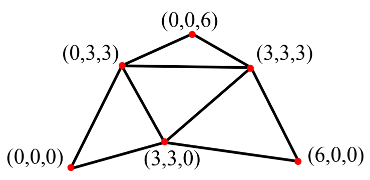
(a) Compute the surface point when $(u, v) = (0, 1)$.

$(0, 6, 4)$.

(b) Compute the surface point when $(u, v) = (0.5, 0.5)$.

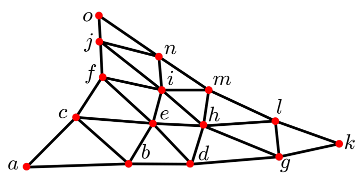
$(4, 3, 1)$.

8. Given the following quadratic Bézier triangle, compute the surface point when $(u, v) = (1/3, 1/3)$.



$(2, 2, 2)$

9. Shown below is the triangular net of a quartic (degree-4) Bézier triangle. When $v = 1$, the Bézier triangle is reduced to k. When $w = 1$, it is o. Define the equation of the curve when $u = 0$.



It is either $(1-v)^4 o + 4v(1-v)^3 n + 6v^2(1-v)^2 m + 4v^3(1-v) l + v^4 k$ or $(1-w)^4 k + 4w(1-w)^3 l + 6w^2(1-w)^2 m + 4w^3(1-w) n + w^4 o$.