[CSE4170: 기초 컴퓨터 그래픽스]

기 말 고 사

담당교수: 임 인 성

- 답은 연습지가 아니라 답안지에 기술할 것. 답안지 공간이 부족할 경우, 답안지 뒷면에 기술하고, 해당 답안지 칸에 그 사실을 명기할 것. 연습지는 수거하지 않음.

1. 다음은 색깔 혼합(Color Blending)에 관한 문제이다. $(c_S\ \alpha_S)$와 $(c_D\ \alpha_D)$를 두 이미지 S와 D의 대응되는 화소의 미리 곱한 색깔(pre-multiplied color)이라 할 때, 두 색깔의 합성을 통하여 생성한 결과 색깔 $(c_O\ \alpha_O)$는 다음과 같이 표현할 수 있다.

$$\begin{pmatrix} c_O \\ \alpha_O \end{pmatrix} = F_S \begin{pmatrix} c_S \\ \alpha_S \end{pmatrix} + F_D \begin{pmatrix} c_D \\ \alpha_D \end{pmatrix}$$

   (a) 만약 결과 값이 $(0.8, 0.0, 0.0, 0.8)$이라 할 때, 이는 그 화소를 어떻게 칠해야 한다는 것인지 정확히 기술하라.

   (b) 그림 1이 암시하는 합성 연산의 경우 해당하는 F_S와 F_D의 값은 각각 얼마인가?

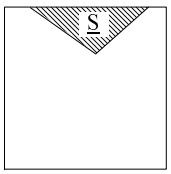


Figure 1: 색깔 혼합 예 1

   (c) 만약 S **over** D 연산을 적용한다면, 합성 후 α_O는 어떤 값을 가질 지를 S와 D의 관련 값을 사용하여 정확히 기술하라.

   (d) 그림 2의 상황을 고려하자. 지금 결과 이미지의 한 화소에 네 개의 물체 Y, R, G, 그리고 K가 순서대로 투영되고 있는데, 이 물체들은 각각 순서대로 '순수한' 노란색, 빨간색, 초록색, 그리고 검정색을 띠고 있으며, 각 물체의 불투명도(opacity)는 각 물체 옆에 기술되어 있다. 지금 이 물체들의 색깔을 앞에서 뒤로 가면서(front-to-back order) **over** 연산을 사용하여 합성한다고 할 때, Y, R, 그리고 G 물체까지 합성하였을 때의 불투명도 값이 무엇일지 기술하라.

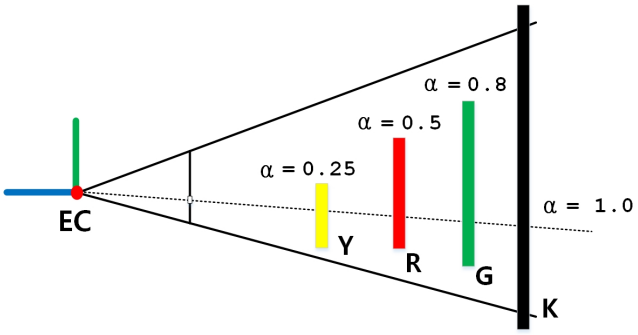


Figure 2: 색깔 합성 예 2

   (e) (바로 위 문제에 이어서) 네 개의 물체 모두에 대한 합성이 끝났을 때, 최종 결과 색깔을 미리 곱한 색깔 형식으로 표현하라.

   (f) 아래에는 투명한 물체들을 그려주는 프로그램의 fragment shader가 주어져 있다. 여기서 uniform variable u_flag_blending이 0, 1, 또는 2 값을 가질 경우, 각각 합성 기능을 사용하지 않거나, 뒤에서 앞으로 가면서 합성을 하거나, 또는 앞에서 뒤로 가면서 합성을 해주게 된다. 또한, 함수 `obj_c(*,*)`는 현재 fragment에 칠할 실제 색깔(미리 곱한 색깔이 아닌)을 계산해주며, 각 물체를 그리기 직전 해당하는 불투명도 값을 uniform variable인 `u_frag_a`에 설정을 해주도록 되어 있다. 만약 u_flag_blending 값을 1로 설정할 경우, 올바른 결과를 얻기 위하여, `glBlendFunc(*,*)` 함수 호출 시 사용할 두 인자를 아래에서 찾아 순서대로 기술하라.

```
GL_ZERO, GL_ONE,
GL_SRC_ALPHA, GL_DST_ALPHA,
GL_ONE_MINUS_SRC_ALPHA,
GL_ONE_MINUS_DST_ALPHA
```

```
========================================
#version 420
    ⋮
uniform int u_flag_blending = 0;
uniform float u_frag_a = 1.0f;
in vec3 v_position_EC;
in vec3 v_normal_EC;
layout (location = 0) out vec4 f_c;

vec3 obj_c(in vec3 P_EC, in vec3 N_EC) {
  vec3 object_color;
    ⋮ // Compute the object's color.
  return object_color;
}
void main(void) {
  vec3 C = obj_c(v_position_EC,
              normalize(v_normal_EC));
  if (u_flag_blending == 0)
     f_c = vec4(C, 1.0f);
  else if (u_flag_blending == 1)
     f_c = vec4( (가) , (나) );
  else if (u_flag_blending == 2)
     f_c = vec4( (다) , u_frag_a);
}
========================================
```

(g) (바로 위 문제에 이어) 이 경우 fragment shader의 (가)와 (나)에 들어갈 내용을 OpenGL Shading Language의 문법에 맞게 정확히 기술하라.

(h) 만약 u_flag_blending 값을 2로 설정할 경우, 이 프로그램이 제대로 작동하기 위하여, glBlendFunc(*,*) 함수 호출에 필요한 두 인자를 순서대로 기술하라.

(i) (바로 위 문제에 이어) 이 경우 fragment shader의 (다)에 들어갈 내용을 OpenGL Shading Language의 문법에 맞게 정확히 기술하라.

(j) 다음과 같은 단순한 fragment shader를 사용하는 색깔 합성 프로그램을 고려하자.

```
========================================
#version 420
uniform vec4 u_fragment_color;
layout (location = 0) out vec4 final_color;
void main(void) {
  final_color = u_fragment_color;
}
========================================
```

아래의 OpenGL 코드에서는 그림 3에 도시한 바와 같이 서로 다른 영역을 가지는 세 개의 사각형을 뒤에서 앞으로 오면서 차례대로 그려주고 있다. 이 프로그램 수행 후 A 화소 지점에 해당하는 color buffer 영역에 저장되어 있는 실제 색깔(미리 곱한 색깔이 아닌) (R, G, B) 값을 기술하라. (주의: RGB 각 채널 값은 0과 1 사이의 값으로 정규화하여 답할 것)

```
========================================
void display(void) {
  glDisable(GL_DEPTH_TEST);
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  glClear(GL_COLOR_BUFFER_BIT);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_DST_ALPHA);
  glUseProgram(h_ShaderProgram_PS);
  glUniform4f(loc_u_fragment_color,
              1.0f, 0.0f, 0.0f, 0.5f);
  draw_Rectangle_2();
  glUniform4f(loc_u_fragment_color,
              1.0f, 0.0f, 0.0f, 0.5f);
  draw_Rectangle_1();
  glUniform4f(loc_u_fragment_color,
              0.0f, 0.0f, 1.0f, 1.0f);
  draw_Rectangle_0();
  glUseProgram(0);
  glDisable(GL_BLEND);
  glEnable(GL_DEPTH_TEST);
  glutSwapBuffers();
}
========================================
```

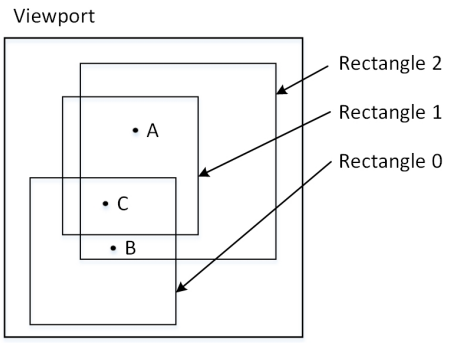


Figure 3: 세 개의 사각형 영역

(k) (바로 위 문제에 이어) 이 프로그램 수행 후 B 화소 지점에 해당하는 color buffer 영역에 저장되어 있는 실제 색깔(미리 곱한 색깔이 아닌) (R, G, B) 값을 기술하라. (주의: RGB 각 채널 값은 0과 1 사이의 값으로 정규화하여 답할 것)

(l) 위 프로그램에서 blending factor를 설정하

는 부분을 다음과 같이 변경할 경우,

```
glBlendFunc(GL_ONE_MINUS_DST_ALPHA,
            GL_ZERO);
```

이 프로그램 수행 후 C 화소 지점에 해당하는 color buffer 영역에 저장되어 있는 실제 색깔(미리 곱한 색깔이 아닌) (R, G, B) 값을 기술하라. (주의: RGB 각 채널 값은 0과 1 사이의 값으로 정규화하여 답할 것)

2. 다음은 카메라의 설정에 관한 문제이다. 그림 4에는 OpenGL Compatibility Profile에서 제공하는 `gluLookAt(*)` 함수의 호출에 대하여 뷰잉변환 행렬을 계산하는 과정(이하 "이 그림")이 주어져 있으며, 그림 5에는 glm::lookAt() 함수에 대한 구현 코드(이하 "이 함수")가 주어져 있는데, 이들을 참조하며 답하라.

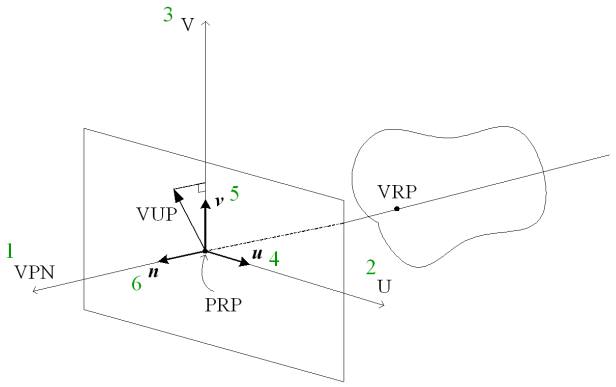


Figure 4: `gluLookAt(*)` 함수를 통한 카메라의 설정

(a) 이 그림에서 이 함수의 인자 중의 하나인 3차원 벡터 `eye`에 해당하는 기호의 이름을 기술하라.

(b) 이 그림의 u, v, 그리고 n 벡터는 각각 카메라를 기준으로 오른쪽, 위쪽, 그리고 바라보는 반대 방향에 대한 단위 벡터(unit vector)들이다. 이 함수의 `Line (a)` 문장 수행 후 벡터 `f`에 저장되는 값을 위의 벡터들을 사용하여 정확히 기술하라.

(c) 문맥 상 이 함수의 `Line (b)`와 `Line (c)` 문장의 `smile(*,*)` 함수는 두 개의 3차원 벡터를 인자로 받아 어떠한 계산을 해줄까?

(d) 이 함수의 `Line (d)` 문장 수행 결과 4행 4열 행렬 `Result`에는 어떤 값이 저장 될까?

(e) 문맥 상 이 함수의 `Line (e)`와 `Line (f)`의 빈곳에 들어갈 내용을 이 프로그램의 문법에 맞게 정확히 기술하라.

3. 이 문제는 어떤 이진트리 구조를 사용하여 이 트리에 저장되어 있는 다각형들을 주어진 카메라 시점을 기준으로 **앞에서 뒤로 가면서 정렬한 순서로 (in front-to-back order) 그려주는** 문제에 관한 것이다. 우선 이 트리는 다음과 같이 정의된다.

```
==================================
typedef struct _Polygon {
   int n_v;
   float *vertex;
   float *normal;
   float plane[4];
} Polygon;

typedef struct _TREE {
   Polygon *polygon;
   struct _TREE *fc, *bc;
} TREE;
==================================
```

(a) 다각형 한 개에 대한 정보를 저장해주는 `Polygon` 타입의 변수의 `plane[4]` 필드는 나머지 세 개의 변수들에 의해 정의되는 다각형을 포함하는 평면의 식 $a \cdot x + b \cdot y + c \cdot z + d = 0$의 네 개의 상수 a, b, c, 그리고 d를 순서대로 저장하고 있는데, 이 다각형의 법선 벡터는 이 평면의 식에 반영이 되어 있다. 만약 이 다각형이 점 $(1, 1, 1)$을 지나고, 법선 벡터가 $(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$이라면, 배열 `plane[4]`에 들어갈 네 개의 값을 순서대로 기술하라.

(b) 다음은 실제로 포인터 변수 `tree`가 가리키는 이진트리에 저장된 다각형들을 앞에서 기술한 순서대로 그려주는 함수이다. 문맥 상 이 함수의 또 다른 인자 `p`는 어떤 값을 저장한 배열에 대한 포인터 변수일까?

```
==================================
void draw_polygons(TREE *tree,
                   float *p) {
  if (tree == NULL) return;
  if (check_side(p, tree->polygon)
          == TREE_FRONT) {
    draw_polygons(tree->fc, p);
    draw_a_polygon(tree->polygon);
    draw_polygons(tree->bc, p);
  }
  else {
        (A)
  }
```

```
namespace glm {
  ...
  template <typename T, precision P> GLM_FUNC_QUALIFIER tmat4x4<T, P>
     lookAt(tvec3<T, P> const &eye, tvec3<T, P> const &center, tvec3<T, P> const &up) {
        tvec3<T, P> const f(normalize(center - eye)); // Line (a)
        tvec3<T, P> const s(normalize(smile(f, up))); // Line (b)
        tvec3<T, P> const u(smile(s, f)); // Line (c)
        tmat4x4<T, P> Result(1); // Line (d)
        Result[0][0] = s.x; Result[1][0] = s.y; Result[2][0] = s.z;
        Result[0][1] = u.x; Result[1][1] = u.y; Result[2][1] = u.z;
        Result[0][2] = -f.x; Result[1][2] = -f.y; Result[2][2] = -f.z;
        Result[3][0] = -dot(s, eye);
        Result[3][1] =               ; // Line (e)
        Result[3][2] =               ; // Line (f)
        return Result;
     }
}
```

Figure 5: glm::lookAt() 함수의 구현

```
    }
    ==================================
```

(c) 다음은 위 문제 코드의 `check_side(*,*)` 함수에 대한 코드인데, `p`가 가리키는 점이 `tree->polygon->plane[4]`가 정의하는 평면의 앞쪽에 있으면 `TREE_FRONT`를, 아니면 `TREE_BACK`을 리턴해주도록 되어 있다(편의상 이 점이 평면 상에 있을 경우 뒤쪽에 있다고 가정함). 이 함수의 (B) 부분에 들어갈 내용을 C/C++ 문법에 맞게 정확히 기술하라.

```
==================================
#define TREE_BACK 0
#define TREE_FRONT 1
int check_side(float *p,
               Polygon *poly) {
   if ( (B) )
      return TREE_FRONT;
   else
      return TREE_BACK;
}
==================================
```

(d) 앞의 코드에서 `draw_polygons(*,*)` 함수가 올바르게 작동하기 위하여 (A) 부분에 들어갈 내용을 C/C++ 문법에 맞게 정확히 기술하라.

(e) 이 이진트리에 저장되어 있는 다각형의 개수가 n이라할 때, `draw_polygons(*,*)` 함수의 시간 복잡도를 Big-O 기호를 사용하여 기술하라.

(f) 이 문제에서 사용한 이진트리의 영문 이름을 기술하라.

4. (바로 위 문제에 이어) 이번 문제도 바로 앞 문제에서 사용한 이진트리의 응용에 관한 문제이다. 지금 가상의 3차원 실내 공간을 자유롭게 돌아다니려 하는데, 사용자가 벽면을 뚫고 지나가지 못하도록 벽면들에 대한 다각형 정보를 저장하고 있는 이 이진트리를 사용하여 다음과 같은 함수 호출을 통하여 충돌검사를 수행 하려한다.

```
LOS(tree, A, B);
```

`LOS(*,*,*)` 함수의 목적은 3차원 공간의 두 점 A와 B에 의해 정의된 선분이 실내 공간의 벽면에 해당하는 임의의 다각형과 교차를 하는지 판별하는 것인데, 이 함수는 다음과 같이 정의할 수 있다.

```
=======================================
int LOS(TREE *T, float *s, float *e) {
  int side_s, side_e, side_f, side_b;

  if (!T) return (A) ;
  side_s = check_side(s, T->polygon);
  side_e = check_side(e, T->polygon);
  if ((side_s == TREE_FRONT)
      && (side_e == TREE_FRONT)) {
```

```
        return LOS( (C) );
    }
    else if ((side_s == TREE_BACK)
        && (side_e == TREE_BACK)) {
       return LOS( (D) );
    }
    else {
     if ( (B) ) return 0;
     else {
       side_f = LOS(T->fc, s, e);
       side_b = LOS(T->bc, s, e);
       return (E) ;
     }
    }
}
=======================================
```

(a) 이 함수는 s와 e가 가리키는 두 점을 연결한 선분이 최소한 한 개의 벽면 다각형과 교차하면 0을 아니면 1을 리턴해주도록 설계되어 있다. 문맥 상 이 코드의 (A)에 들어갈 상수 값을 기술하라.

(b) 문맥 상 (B) 부분의 조건식은 언제 TRUE 값을 가지게 될지 정확히 기술하라.

(c) 문맥 상 (C)에 들어갈 내용을 C/C++ 언어 문법에 맞게 정확히 기술하라.

(d) 문맥 상 (D)에 들어갈 내용을 C/C++ 언어 문법에 맞게 정확히 기술하라.

(e) 문맥 상 (E)에 들어갈 내용을 C/C++ 언어 문법에 맞게 정확히 기술하라.

5. 그림 6은 자동차 차체 모델을 기준으로 회전변환 R과 이동변환 T를 통하여 운전석에 카메라를 배치한 후, 강체변환 M_B를 통하여 차체를 세상 좌표계에 배치하는 모습을 도시하고 있다. 이때의 뷰잉변환 행렬 M_V를 위의 세 개의 4행 4열 행렬을 사용하여 표현하라.

6. 그림 7(a)에는 두 가지 형태의 퐁의 조명 모델 식이 주어져 있고, (b)-(d)에는 OpenGL 시스템에서 사용하는 기본 조명 공식이 기술되어 있다.

(a) (a)의 공식에서 H_i를 이 공식의 다른 기호를 사용하여 수학적으로 정확히 표현하라.

(b) (a)의 공식에서 Lambert's cosine law를 표현해주는 부분만 정확히 기술하라.

(c) (a)의 공식에서 만약 평행 광원과 평행 투영을 사용할 경우, 물체의 위치와 방향이 바뀌었을 때 영향을 받는 변수를 모두 나열하라 (d_i는 제외).

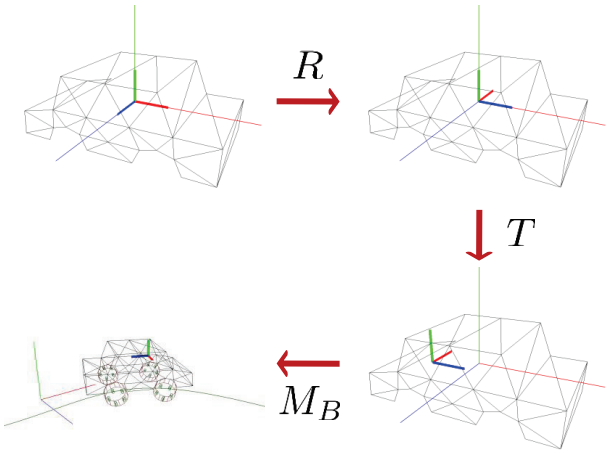


Figure 6: 자동차 운전석에 대한 카메라의 배치

(d) (a)의 공식에서 카메라에서 바라보는 방향에 직접적으로 영향을 받는 변수를 모두나열하라.

(e) (a)의 공식에서 만약 점 광원을 사용할 경우 광원의 위치의 변화에 영향을 받는 변수를 모두 나열하라.

(f) (a)의 공식에서 현재 퐁의 조명 모델 식이 적용되고 있는 물체 표면 지점의 좌표를 P라고 하자. (b)의 공식에서 이 P에 해당하는변수 (또는 수식)를 기술하라.

(g) (b)의 공식에서 (a)의 공식의 d_i에 해당하는 변수 (또는 수식)를 기술하라.

(h) (b)의 공식에서 $\odot$은 주어진 두 벡터 $\mathbf{u}$와 $\mathbf{v}$에 대하여 어떠한 값을 계산해주는 연산자인지 정확히 기술하라.

(i) (b)의 공식에서 $\mathbf{0}$ 벡터가 아닌 임의의 벡터 $\mathbf{v}$에 대해 $\hat{\mathbf{v}}$는 무엇을 의미하는가?

(j) 정반사 방향 (reflection direction), 즉 정반사 물질이 입사 광선을 가장 강하게 반사시키는 방향을 (b) 공식의 변수들을 사용하여 표현하라. ($\mathbf{n} \odot \overrightarrow{\mathbf{VP}}_{pli}$은 양수가 가정함)

(k) (b)의 공식에서 정반사로 인한 하이라이트 영역의 크기를 조절할 수 있는 변수를 기술하라.

(l) (b)의 공식에서 간접 조명 (indirect illumination) 효과를 (근사적이나마) 생성하는데 필요한 간접적으로 들어오는 빛, 즉 간접적인 입사 광선과 직접적인 연관성이 있는 변수들을 모두 기술하라.

(m) (c)의 공식의 조건식 $\mathbf{P}_{pli}$'s $w \neq 0$이 만족되는 상황은 어떤 경우인지 정확히 기술하라.

(n) (d)의 공식에서 문맥상 (A)와 (B)에 공통으로 들어갈 수식을 정확히 기술하라.

$$I_\lambda = I_{a\lambda} \cdot k_{a\lambda} + \sum_{i=0}^{m-1} f_{att}(d_i) \cdot I_{l_i\lambda} \cdot \{k_{d\lambda} \cdot (N \circ L_i) + k_{s\lambda} \cdot (R_i \circ V)^n\} \quad (1)$$

$$I_\lambda = I_{a\lambda} \cdot k_{a\lambda} + \sum_{i=0}^{m-1} f_{att}(d_i) \cdot I_{l_i\lambda} \cdot \{k_{d\lambda} \cdot (N \circ L_i) + k_{s\lambda} \cdot (N \circ H_i)^n\} \quad (2)$$

(a) 퐁의 조명 모델 식의 두 예

$$\mathbf{c} = \mathbf{e}_{cm} + \mathbf{a}_{cm} * \mathbf{a}_{cs} + \sum_{i=0}^{n-1} (att_i)(spot_i)[\mathbf{a}_{cm} * \mathbf{a}_{cli} + (\mathbf{n} \odot \overrightarrow{\mathbf{VP}}_{pli})\mathbf{d}_{cm} * \mathbf{d}_{cli} + (f_i)(\mathbf{n} \odot \hat{\mathbf{h}}_i)^{s_{rm}}\mathbf{s}_{cm} * \mathbf{s}_{cli}]$$

(b) OpenGL 시스템의 조명 공식

$$att_i = \begin{cases} \frac{1}{k_{0i} + k_{1i}\|\mathbf{VP}_{pli}\| + k_{2i}\|\mathbf{VP}_{pli}\|^2}, & \mathbf{P}_{pli}\text{'s } w \neq 0, \\ 1.0, & \text{otherwise} \end{cases}$$

(c) 빛의 감쇠 효과의 계산

$$spot_i = \begin{cases} (\overrightarrow{\mathbf{P}_{pli}\mathbf{V}} \odot \hat{\mathbf{s}}_{dli})^{s_{rli}}, & c_{rli} \neq 180.0 \text{ and } (\mathtt{A}) \geq (\mathtt{B}), \\ 0.0, & c_{rli} \neq 180.0 \text{ and } (\mathtt{A}) < (\mathtt{B}), \\ 1.0, & c_{rli} = 180.0 \end{cases}$$

(d) 스폿 광원 효과의 계산

Figure 7: 조명 모델 공식 예

7. 그림 8의 OpenGL 렌더링 파이프라인을 보면서 답하라. 이 문제의 질문은 모두 '정상적인 렌더링 과정'을 가정한다. **(어떤 지점을 지칭할 때에는 해당 상자의 기호 ((A)부터 (I)까지)를 사용할 것)**

   (a) 이 그림에서 `out vec3 v_position_EC;` 문장과 `in vec3 v_position_EC;` 문장에서 정의된 동일한 이름의 두 변수간에 직접적으로 정보가 전달되는 지점은?

   (b) '정규화된 3차원 필름'이 정의되는 좌표계는 어느 상자와 어느 상자 사이의 지점에 해당할까?

   (c) 거리 값에 기반을 둔 '은면 제거(hidden surface removal)' 계산이 수행되는 지점은?

   (d) 화면의 윈도우 영역 중 실제로 그림이 그려지는 영역을 결정하는 계산이 수행되는 지점은?

   (e) `glDrawArrays(GL_TRIANGLES, 0, 900);` 문장에서 상수 `GL_TRIANGLES`이 가장 직접적으로 영향을 미치는 상자의 기호는?

   (f) 비로소 CC (Clip Coordinate)의 꼭지점을 NDC (Normalized Device Coordinate)의 꼭지점으로 변환해주는 상자의 기호는?

   (g) 꼭지점의 좌표를 (x, y, z, w)라 할 때 보편적인 렌더링 과정 중 w 값이 1이 아닌 값을 가질 수 있는 지점을 모두 기술하라.

   (h) 사용자가 설정한 카메라의 위치와 방향 정보가 반영되는 지점은?

   (i) 보편적인 렌더링 과정 중 mipmapping 과정이 가장 빈번하게 수행되는 지점은?

   (j) Polygonal shading model 중의 하나인 Gouraud shading 기법을 구현할 때, 실제 Phong의 ilumination model 식이 적용되는 상자의 기호는?

   (k) `glBlendFunc(GL_SRC_ALPHA, GL_DST_ALPHA);` 문장이 직접적으로 영향을 미치는 계산이 수행되는 지점은?

   (l) 보편적인 렌더링 과정에서 다음 기하변환

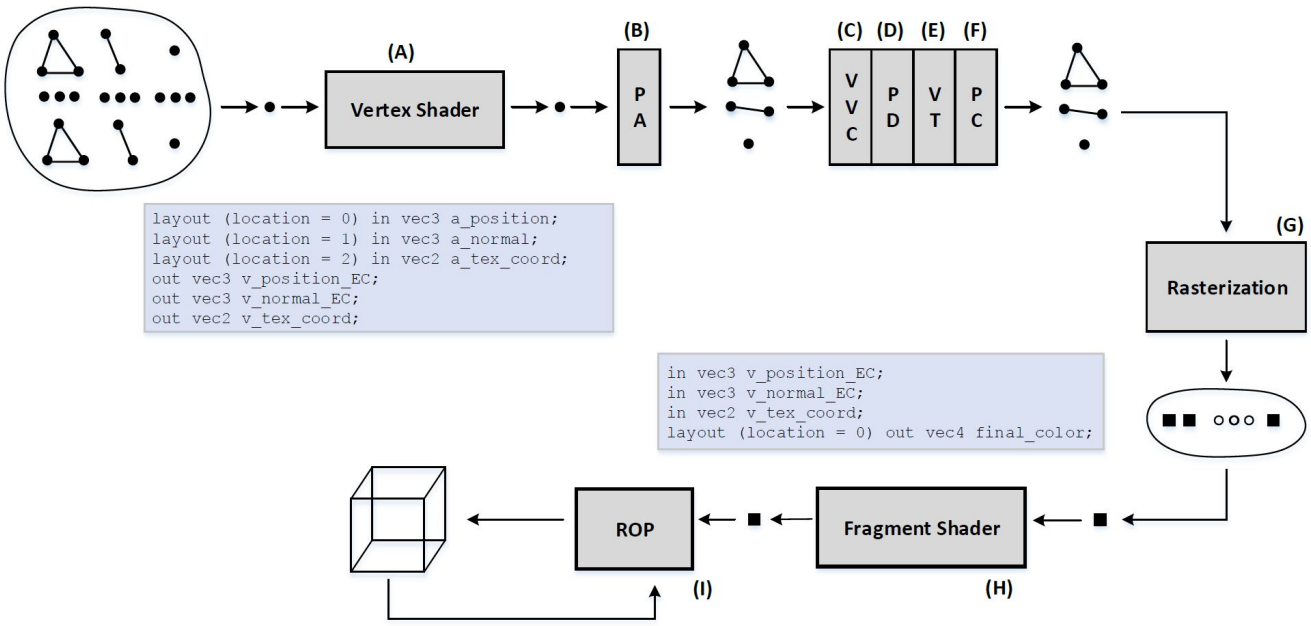


Figure 8: OpenGL 렌더링 파이프라인

행렬과 가장 관련이 높은 상자의 번호는?

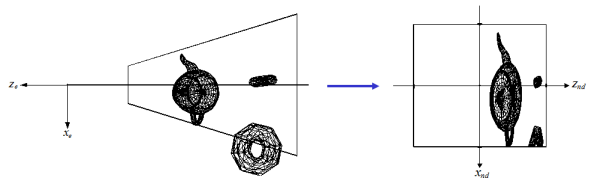
$$\begin{bmatrix} \frac{cot(\frac{fovy}{2})}{asp} & 0 & 0 & 0 \\ 0 & cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

(m) 위의 행렬에서 f 값에 의해 정의되는 평면 뒤쪽에 있는 기하 물체가 화면에서 사라지게 되는 지점은?

(n) 삼각형의 꼭지점의 나열 순서를 통하여 기하 물체의 일부를 제거할 수 있는 지점은?

(o) `glFrontFace(GL_CW);` 문장이 가장 직접적으로 영향을 미치는 지점은?

(p) 다음 그림이 암시하는 렌더링 계산과 가장 직접적으로 관련이 있는 두 상자의 기호를 기술하라.



8. 다음은 텍스춰 필터링에 관한 문제이다. 아래 그림을 보면서 답하라.

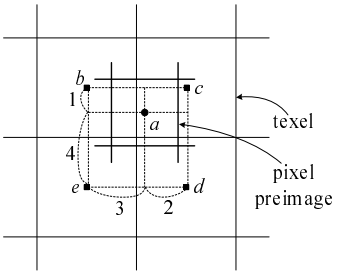
(a) 레벨 4의 밉맵 텍스춰의 한 텍셀이 나타내는 영역의 면적은 레벨 0의 밉맵 텍스춰의 한 텍셀이 나타내는 영역의 면적의 몇 배일까?

(b) 해상도가 512 × 512이고 각 텍셀 당 `GL_UNSIGNED_BYTE`와 `GL_RGBA` 형식을 사용하는 텍스춰 이미지에 대하여 모든 가능한 레벨에 대하여 밉맵을 구성할 경우 총 몇 바이트의 텍스춰 메모리가 필요할까?

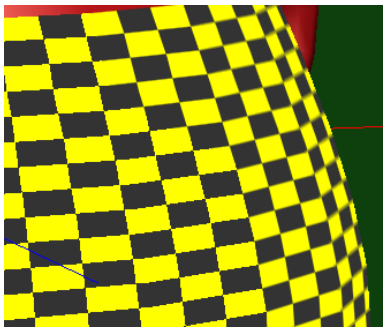
(c) 그림 9(a)에서 a는 픽셀에 대한 원상(pre-image)의 중점을 가리키고, b부터 e까지는 a를 포함하는 주변 텍셀의 중심점을 나타낸다. b, c, d, 그리고 e 지점의 텍셀 색깔이 각각 $(1,1,1)$, $(1,0,0)$, $(1,1,0)$, 그리고 $(0,1,1)$이라고 하자. 각각 최근 필터와 선형 필터를 사용할 경우 a 지점에 대하여 계산되는 색깔은 무엇일까? 이 그림에서 숫자는 거리에 대한 비율을 나타내고 있음.

(d) 그림 9(b)는 확대 필터는 `GL_NEAREST`, 그리고 축소 필터는 `GL_LINEAR_MIPMAP_LINEAR`를 사용한 경우의 렌더링 결과이다. 과연 이 그림의 상황은 축소 상황이 발생한 것인지, 아니면 확대 상황이 발생한 것인지 구체적으로 기술하라.

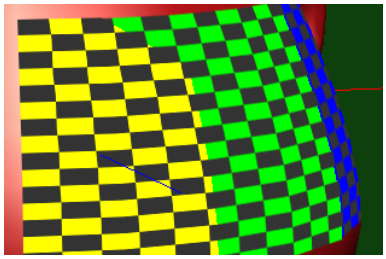
(e) 그림 9(c)는 축소 상황이 발생한 경우이다. 서로 다른 레벨의 밉맵 텍스춰에 대해서 서로 다른 색깔의 텍스춰 이미지를 사용하였는데, 과연 이 그림은 삼선형 필터(trilinear



(a) 텍셀 값의 계산



(b) 확대와 축소 현상



(c) 삼선형 필터의 적용

Figure 9: 텍스춰 필터링

interpolation filter)를 사용한 결과라 할 수 있는가? 예/아니오로 답하고 그렇게 답한 이유를 밝혀라.

9. 본 시험 문제지의 마지막 쪽에는 OpenGL Shading Language (GLSL)를 사용하여 가급적 그림 7의 퐁의 조명 모델 식에 기반을 두어 퐁 쉐이딩 방법을 구현한 버텍스 쉐이더와 프래그먼트 쉐이더 프로그램이 주어져 있다.

   (a) 보편적인 렌더링 상황에서, 어떠한 조건을 만족할 경우 5번 문장의 행렬의 왼쪽-위 3행 3열 행렬과 6번 문장 행렬의 내용이 동일할까?

   (b) 125번 문장에서 `corrected_normal_EC` 벡터를 정규화를 해주고 있는데, (정확한 계산을 위해서) 이 과정이 필요한 이유는 무엇일까?

   (c) OpenGL 렌더링 과정중 view volume clipping 계산이 수행되는 좌표계에서의 좌표값을 가지는 변수를 모두 기술하라.

   (d) GPU의 메모리 영역에 사용자가 할당한 Vertex Buffer Object의 데이터와 직접적으로 연관이 있는 변수를 모두 기술하라.

   (e) 현재 처리하고 있는 광원이 점광원인지는 어떻게 판별하고 있는가?

   (f) 점 광원을 처리할 때 광원과 쉐이딩하려는 지점 간의 거리가 계산이 되는 문장의 번호를 기술하라.

   (g) 평행 광원을 처리할 때 빛의 감쇠 효과는 어떻게 처리가 될까?

   (h) 그림 7(c)의 i번째 광원에 해당하는 k_{2i} 값을 저장하고 있는 변수 (또는 식)를 GLSL 문법에 맞게 정확히 기술하라.

   (i) 100번 문장 시점에서 그림 7(a)의 조명 공식 (1)에서 사용한 벡터 V 값을 이 쉐이더 코드의 문맥에서 GLSL 문법에 맞게 정확히 기술하라.

   (j) 92번 문장의 (가) 부분에 들어갈 내용을 GLSL 문법에 맞게 정확히 기술하라.

   (k) 이 쉐이더 코드에서는 halfway vector를 사용하여 쉐이딩을 하고 있다. 101번 문장의 (나) 부분에 들어갈 수식을 GLSL 문법에 맞게 정확히 기술하라.

   (l) (위 문제에 이어서) 102 문장의 (다) 부분에 들어갈 수식을 GLSL 문법에 맞게 정확히 기술하라.

   (m) 이 쉐이더 코드의 68번 문장에서 93번 문장의 내용 중 명확히 틀린 부분이 있다. 문장 번호와 함께 틀린 부분을 명확히 밝히고 어떻게 고쳐야며 할지 설명하라.

   (n) 108 문장의 (라) 부분에 들어갈 변수나 수식을 GLSL 문법에 맞게 정확히 기술하라.

   (o) 그림 7(b)의 조명 식의 변수 f_i의 사용 목적이 이 쉐이더에서는 어떻게 달성이 되는지 관련 문장 번호를 지칭하면서 정확히 설명하라.

```
/*------  Vertex Shader  -------*/
#version 430

uniform mat4 u_ModelViewProjectionMatrix;
uniform mat4 u_ModelViewMatrix;
uniform mat3 u_ModelViewMatrixInvTrans;

layout(location = 0) in vec3 a_position;
layout(location = 1) in vec3 a_normal;
layout(location = 2) in vec2 a_tex_coord;
out vec3 v_position_EC;
out vec3 v_normal_EC;
out vec2 v_tex_coord;

void main(void) {
    v_position_EC = vec3(u_ModelViewMatrix*vec4(a_position, 1.0f));
    v_normal_EC = normalize(u_ModelViewMatrixInvTrans*a_normal);
    v_tex_coord = a_tex_coord;

    gl_Position = u_ModelViewProjectionMatrix*vec4(a_position, 1.0f);
}

/*------  Fragment Shader  -------*/
#version 430

struct LIGHT {
    vec4 position;
    vec4 ambient_color, diffuse_color, specular_color;
    vec4 light_attenuation_factors;
    vec3 spot_direction;
    float spot_exponent;
    float spot_cutoff_angle;
    bool light_on;
};

struct MATERIAL {
    vec4 ambient_color;
    vec4 diffuse_color;
    vec4 specular_color;
    vec4 emissive_color;
    float specular_exponent;
};

uniform vec4 u_global_ambient_color;
#define NUMBER_OF_LIGHTS_SUPPORTED 4
uniform LIGHT u_light[NUMBER_OF_LIGHTS_SUPPORTED];
uniform MATERIAL u_material;
uniform sampler2D u_base_texture;
uniform bool u_flag_texture_mapping = true;

const float zero_f = 0.0f;
const float one_f = 1.0f;

in vec3 v_position_EC;
in vec3 v_normal_EC;
in vec2 v_tex_coord;
layout(location = 0) out vec4 final_color;

vec4 lighting_equation_textured(in vec3 P_EC, in vec3 N_EC, in vec4 base_color) {
    vec4 color_sum;
    float local_scale_factor, tmp_float;
    vec3 L_EC;

    color_sum = u_material.emissive_color + u_global_ambient_color * base_color;
    for (int i = 0; i < NUMBER_OF_LIGHTS_SUPPORTED; i++) {
        if (!u_light[i].light_on) continue;
        local_scale_factor = one_f;
        if (u_light[i].position.w != zero_f) {
            L_EC = u_light[i].position.xyz - P_EC.xyz;
            if (u_light[i].light_attenuation_factors.w != zero_f) {
                vec4 tmp_vec4;
                tmp_vec4.x = one_f;
                tmp_vec4.z = dot(L_EC, L_EC);
                tmp_vec4.y = sqrt(tmp_vec4.z);
                tmp_vec4.w = zero_f;
                local_scale_factor = one_f / dot(tmp_vec4, u_light[i].light_attenuation_factors);
            }
            L_EC = normalize(L_EC);
            if (u_light[i].spot_cutoff_angle < 180.0f) { // [0.0f, 90.0f] or 180.0f
                float spot_cutoff_angle = clamp(u_light[i].spot_cutoff_angle, zero_f, 90.0f);
                vec3 spot_dir = normalize(u_light[i].spot_direction);

                tmp_float = dot(-L_EC, spot_dir);
                if (tmp_float < cos(radians(u_light[i].spot_cutoff_angle)))
                    tmp_float = pow(tmp_float, u_light[i].spot_exponent);
                else
                    tmp_float = zero_f;
                local_scale_factor *= tmp_float;
            }
        }
        else {
            L_EC = normalize(  (가)  );
        }
        if (local_scale_factor > zero_f) {
            vec4 local_color_sum = u_light[i].ambient_color * u_material.ambient_color;

            tmp_float = dot(N_EC, L_EC);
            if (tmp_float > zero_f) {
                local_color_sum += u_light[i].diffuse_color*base_color*tmp_float;

                vec3 H_EC = normalize(  (나)  );
                tmp_float = dot(  (다)  );
                if (tmp_float > zero_f) {
                    local_color_sum += u_light[i].specular_color
                        *u_material.specular_color*pow(tmp_float, u_material.specular_exponent);
                }
            }
            color_sum += (라)*local_color_sum;
        }
    }
    return color_sum;
}

void main(void) {
    vec3 corrected_normal_EC;
    vec4 base_color;

    if (gl_FrontFacing) corrected_normal_EC = v_normal_EC;
    else corrected_normal_EC = -v_normal_EC;

    if (u_flag_texture_mapping)
        base_color = texture(u_base_texture, v_tex_coord);
    else
        base_color = u_material.diffuse_color;
    final_color = lighting_equation_textured(v_position_EC, normalize(corrected_normal_EC), base_color);
}

```