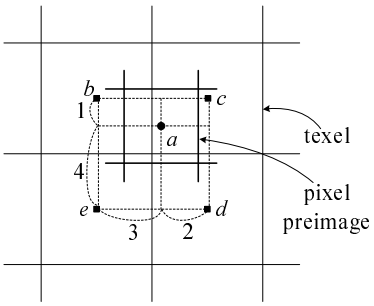
[CSE4170: 기초 컴퓨터 그래픽스]

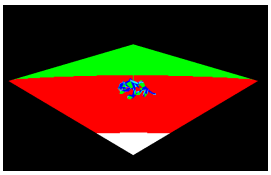
기 말 고 사

담당교수: 임 인 성

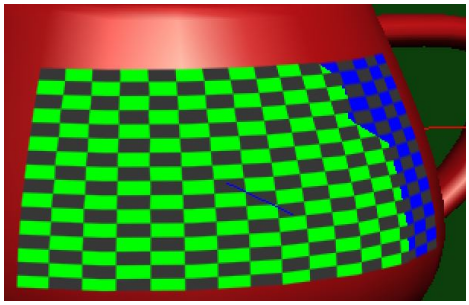
- 답은 연습지가 아니라 답안지에 기술할 것. 답안지 공간이 부족할 경우, 답안지 뒷면에 기술하고, 해당 답안지 칸에 그 사실을 명기할 것. 연습지는 수거하지 않음.



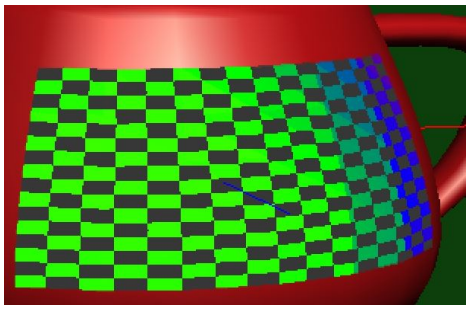
(a) 텍셀 색깔의 계산



(b) 밉맵 레벨의 결정



(c) 텍스춰 필터의 적용 결과 1



(d) 텍스춰 필터의 적용 결과 2

Figure 1: 텍스춰 필터링

1. 다음은 텍스춰 필터링에 관한 문제이다. 그림 1을 보면서 답하라.

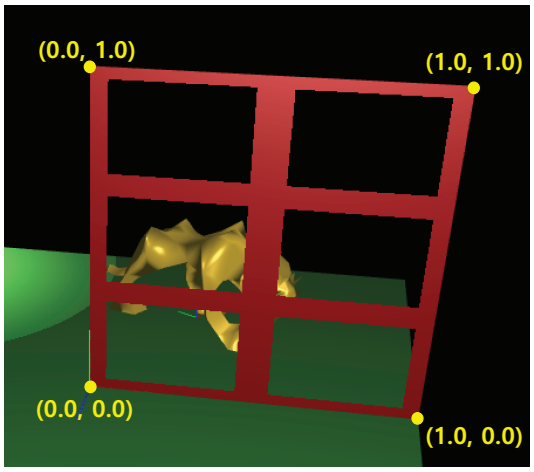
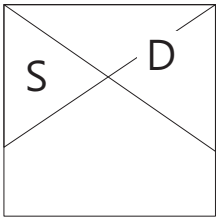
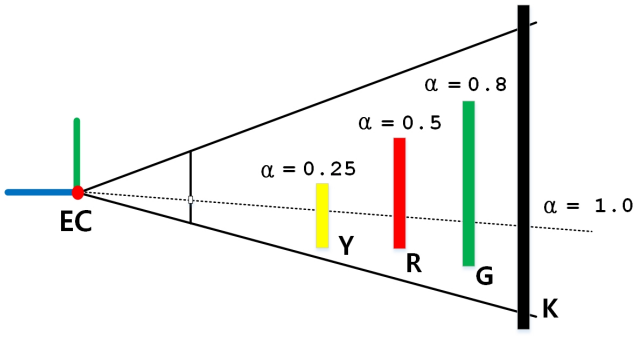


Figure 2: 스크린 효과 생성



(a) 색깔 혼합 예 1



(b) 색깔 혼합 예 2

Figure 3: OpenGL을 사용한 색깔 혼합

(a) 레벨 0의 밉맵 텍스춰의 한 텍셀이 나타내는 영역의 면적은 레벨 3의 밉맵 텍스춰의 한 텍셀이 나타내는 영역의 면적의 몇 배일까?

(b) 텍셀당 두 바이트를 사용하는 해상도가 1024×1024인 레벨 0의 텍스처에 대해 가능한 모든 레벨까지 밉맵을 구성할 경우 몇 바이트의 메모리 공간이 필요할까?

(c) 그림 1(a)에서 a는 픽셀에 대한 원상(pre-image)의 중점을 가리키고, b부터 e까지는 a를 포함하는 주변 텍셀의 중심점을 나타낸다. GL_LINEAR가 의미하는 텍스춰 필터링을 적용할 때 결과로 구해지는 색깔에서 d 텍셀의 색깔이 차지하는 비율을 기술하라.

(d) 위 문제에서 GL_NEAREST가 의미하는 텍스춰 필터링을 적용할 때의 비율을 기술하라.

(e) 그림 1(b)에는 바닥에 대한 사각형 (삼각형 두 개로 구성)에 한 개의 텍스춰를 매핑한 후 밉매핑 기능을 사용할 때 적용되는 레벨 값으로 구역을 나눈 상태를 도시하고 있다. 현재 세 영역으로 구성이 되어 있는데, 어떤 순서대로 레벨 값이 높을지, 그리고 그 이유가 무엇인지 정확히 기술하라.

(f) 그림 1(c)와 (d)는 축소 상황에서 각각 서로 다른 (반대가 되는) 텍스춰 필터링 방법을 적용한 렌더링 결과이다. 과연 두 텍스춰 필터링 방법에 어떤 차이가 있는지 각 방법을 두 가지 방향에서 정확히 기술하라.

2. 다음은 스크린 효과 생성을 위한 OpenGL 프래그먼트 쉐이더 코드의 일부이다.

```
=======================================
   ⋮
in vec2 v_pos_sc;
   ⋮
void main(void) {
 if (screen_effect) {
  float x_mod, y_mod;
  x_mod = mod( (A) , 1.0f);
  y_mod = mod( (B) , 1.0f);
  if ( (x_mod > 0.1f) && (x_mod < 0.9f)
   && (y_mod > 0.1f) && (y_mod < 0.9f) )
   discard;
  }
 final_color = l_e(v_pos_EC,
                   normalize(v_norm_EC));
}
=======================================
```

(a) 그림 2의 스크린 사각형의 각 꼭지점에는 v_pos_sc = (v_pos_sc.x, v_pos_sc.y) 속성으로 설정한 값이 주어져 있다. 이 프로그램이 올바르게 작동하기 위하여 (A)와 (B)에 들어갈 내용을 OpenGL Shading Language 문법에 맞게 정확히 기술하라. (참고: $mod(x, y)$ returns $x - y \cdot floor(\frac{x}{y})$.)

(b) (위 문제를 정확히 해결하였다는 가정하에) 만약 두 번째 if 문장을 다음과 같이 바꿀 경우,

```
if ( (x_mod < 0.1f) || (x_mod >
0.9f) || (y_mod < 0.1f) || (y_mod >
0.9f) )
```

그림 2의 스크린 효과가 어떻게 바뀔지 기술하거나 그림으로 설명하라.

3. 다음은 색깔 혼합 (Color Blending)에 관한 문제이다. (c_S α_S)와 (c_D α_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 α_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) 어떤 미리 곱한 색깔이 $(0.8, 0.8, 0.8, 0.8)$이라면, 이는 그 화소를 어떻게 칠해야 한다는 것인지 정확히 기술하라.

(b) 그림 3(a)가 암시하는 합성 연산의 경우 해당하는 F_S와 F_D의 값은 각각 얼마인가?

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

(d) 그림 3(b)의 상황을 고려하자. 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, Y와 R 물체까지 합성하였을 때의 불투명도 값이 무엇일지 기술하라.

(e) (바로 위 문제에 이어서) 이 물체들의 색깔을 뒤에서 앞으로 가면서 (back-to-front order) **over** 연산을 사용하여 합성한다고 할 때, K, G 그리고 R 물체까지 합성하였을 때의 색깔이 무엇인지 미리곱한 색깔 형식으로 기술하라.

(f) 다음은 투명한 물체들을 그려주는 어떤 프로그램의 프래그먼트 쉐이더 코드이다. 여기서 유니폼 변수 u_flag_blending이 0, 1, 또는 2 값을 가질 경우, 각각 합성 기능을 사용하지 않거나, 뒤에서 앞으로 가면서 **over** 연산을 사용하여 합성을 하거나, 또는 앞에서 뒤로 가면서 **over** 연산을 사용하여 합성을 해주게 된다. 또한, 함수 obj_C(*,*)는 현재 프래그먼트에 칠할 실제 색깔(미리 곱한 색깔이 아닌)을 계산해주며, 불투명도 값은 프래그먼트 속성 변수 frag_a로 들어온다. 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;
in vec3 v_position_EC;
in vec3 v_normal_EC;
in float frag_a;
layout (location = 0) out vec4 f_c;

vec3 obj_C(in vec3 P_EC, in vec3 N_EC) {
  vec3 object_color;
   ⋮ // Compute the fragment's RGB 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) , frag_a);
  else if (u_flag_blending == 2)
     f_c = vec4( (나) , (다) );
}
========================================
```

(g) (바로 위 문제에 이어서) 이 경우 위 코드의 (가)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 이어서) 이 경우 위 코드의 (나)와 (다)에 들어갈 내용을 OpenGL Shading Language의 문법에 맞게 정확히 기술하라.

(j) 다음과 같은 단순한 프래그먼트 쉐이더를 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 4에 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 (R,G,B,A) 값을 기술하라. 이 때 각 색깔 채널 값은 0과 1 사이의 값으로 정규화하여 분수로 답하라. (참고: 각 사각형을 구성하는 프래그먼트의 색깔은 대응하는 glUniform4f() 함수로 설정한 색깔로 설정이 됨)

```
========================================
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();
}
```

```
=======================================
```

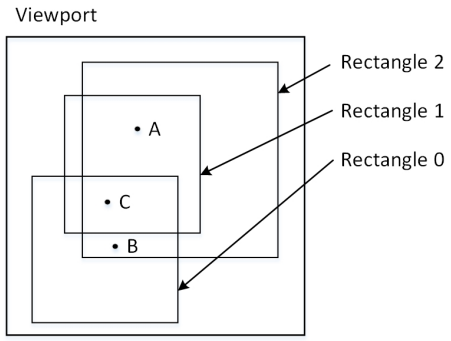


Figure 4: 세 개의 사각형 영역

(k) 위 프로그램에서 blending factor를 설정하는 부분을 다음과 같이 변경할 경우,

```
glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,
            GL_ONE);
```

이 프로그램 수행 후 A 화소 지점에 해당하는 color buffer 영역에 저장되어 있는 (R,G,B,A) 값을 기술하라. 이 때 각 색깔 채널 값은 0과 1 사이의 값으로 정규화하여 분수로 답하라.

(l) (바로 위 문제에 이어서) 이 프로그램 수행 후 B 화소 지점에 해당하는 color buffer 영역에 저장되어 있는 (R,G,B,A) 값을 기술하라. 이 때 각 색깔 채널 값은 0과 1 사이의 값으로 정규화하여 분수로 답하라.

4. 다음은 안개 효과 생성을 위한 OpenGL 프래그먼트 쉐이더 코드의 일부이다. 아래의 코드와 그림 5를 보면서 답하라.

```
=======================================
fog_factor = (far_dist - length(Pe.xyz))
             /(far_dist - near_dist);
fog_factor = clamp(fog_factor, 0.0f,
              1.0f);
final_color = mix(fog_color,
              shaded_color, fog_factor);
=======================================
```

(a) 이 코드는 OpenGL의 눈좌표계(EC)에서 계산을 하고 있다. 문맥 상 그림 5의 한 점 P_e 지점에서의 안개 효과 생성을 위하여 d_0와 d_1 중 어떤 거리 척도를 사용하고 있는가?

(b) d_0와 d_1 중 위 코드에서 사용하고 있는 것이 아닌 다른 척도를 사용하려면, 첫 번째 문장을 어떻게 수정을 해야할지, OpenGL Shading Language의 문법에 맞게 기술하라. (참고: $0 <$ `near_dist` $<$ `far_dist`)

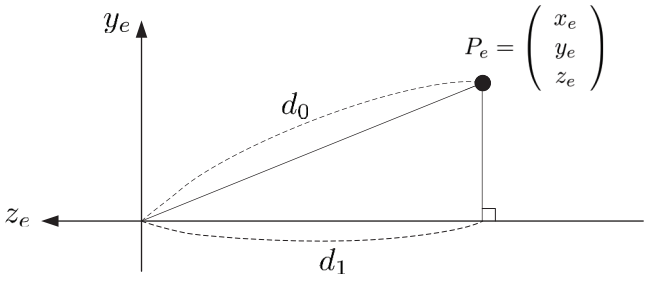


Figure 5: 안개 효과 생성

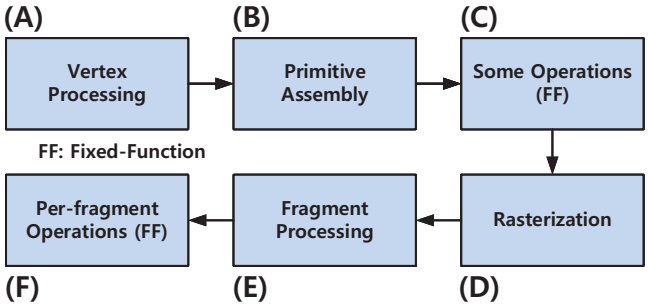


Figure 6: 실시간 렌더링 파이프라인

(c) `mix()` 함수는 선형 보간에 기반을 두고 있다. 다음과 같은 변수 값에 대해,

`fog_color` $= (0.8, 0.8, 0.8, 1.0)$,
`shaded_color` $= (0.4, 0.2, 0.8, 1.0)$,
`fog_factor` $= 0.4$,

`final_color`의 결과 값을 기술하라.

5. 다음은 실시간 렌더링 파이프라인에 관한 문제이다. 그림 6의 렌더링 파이프라인을 보면서 한 학기 동안 수업에서 배운 OpenGL 렌더링 파이프라인을 바탕으로 답하라. 아래의 문제에서 별도로 기술하지 않으면 '정상적인 OpenGL 렌더링' 과정을 가정한다. (필요할 경우 NDC, CC, WdC, MC, EC, 그리고 WC 등의 기호를 사용하여 답할 것)

(a) '정규화된 3차원 필름'을 기준으로 설정된 좌표계가 존재하는 박스의 기호를 기술하라.

(b) 투명한 물체 효과를 생성하기 위하여 색깔을 섞어주는 계산을 가장 자연스럽게 수행할 수 있는 지점에 대한 박스의 기호를 기술하라.

(c) GL_LINEAR라는 OpenGL 인자가 직접적으로 적용이 되는 박스의 기호를 기술하라.

(d) Gouraud 쉐이딩 계산 시 자연스럽게 lighting equation이 적용이 되는 지점에 대한 박스의 기호를 기술하라.

(e) 삼각형을 구성하는 꼭지점들의 회전 방향을 통하여 그 삼각형의 앞면이 보이는지 뒷면이 보이는지를 판별하는 지점이 존재하는 박스의 기호를 기술하라.

(f) 계층적 모델링을 위한 기하 변환이 빈번하게 일어나는 박스의 기호를 기술하라.

(g) (A) 박스는 OpenGL의 어느 좌표계에서 어느 좌표계까지의 기하 계산을 담당하는가?

(h) 꼭지점 속성 값이 프래그먼트로 전달이 되는 지점에 대한 박스의 기호를 기술하라.

(i) (C) 박스는 OpenGL의 어느 좌표계에서 어느 좌표계까지의 기하 계산을 담당하는가?

(j) 구조적으로 프레임 버퍼를 가장 빈번하게 접근해야하는 박스의 기호를 기술하라.

(k) 뷰폿 변환이 수행이 되는 지점이 존재하는 박스의 기호를 기술하라.

(l) 뷰잉 변환이 수행이 되는 지점이 존재하는 박스의 기호를 기술하라.

(m) 원근 투영시 원근감이 생성이 되는 지점이 존재하는 박스의 기호를 기술하라.

(n) 카메라를 기준으로 설정된 좌표계가 존재하는 박스의 기호를 기술하라.

(o) `glm::perspective()` 함수가 설정한 3차원 영역을 벗어난 기하 프리미티브들을 제거하는 계산이 수행되는 박스의 기호를 기술하라.

(p) 픽셀 단위의 안개 효과를 가장 자연스럽게 생성할 수 있는 지점에 대한 박스의 기호를 기술하라.

6. 다음은 어떤 버텍스 쉐이더 코드의 일부인데, 여기서 `Pos`, `pos`, `Norm`, 그리고 `norm`은 모두 `vec3` 타입의 변수들이다.

```
  ⋮
void main(void) {
  gl_Position = M_t*vec4(pos, 1.0f);
  Pos = vec3(M_p*vec4(pos, 1.0f));
  Norm = normalize(M_n*norm);
}
```

(a) 마지막 두 문장은 좌표와 법선 벡터 방향이 각각 `pos`와 `norm`인 점에 대한 아핀 변환을 수행하고 있다. 보편적인 렌더링 상황에서 `M_p` 행렬이 다음과 같을 때, `M_n`은 어떤 행렬이어야 하는지 정확히 기술하라. (주의: 행과 열의 개수가 정확해야 함)

$$M_p = \begin{bmatrix} 0 & 0 & -1 & 2 \\ 0 & 1 & 0 & -3 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) `norm` 변수에는 항상 길이가 1인 법선 벡터 값이 저장이 되어 있다고 가정하자. 만약 `M_p`가 위 문제와 같을 때 과연 마지막 문장에서의 `normalize()` 함수 호출이 필요한지 아니면 불필요한지를 정확한 수학적인 개념을 사용하여 기술하라.

(c) 만약 `M_p` 행렬이 다음과 같을 때, `M_n`은 어떤 행렬이어야 하는지 정확히 기술하라. (주의: 행과 열의 개수가 정확해야 함)

$$M_p = \begin{bmatrix} 0 & 0 & -2 & 1 \\ 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & 6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(d) 이 코드에서 래스터화 과정 중 프래그먼트 쉐이더로 전달이 되는 정보를 저장하고 있는 변수들을 모두 기술하라.

(e) 이 코드에서 버텍스 쉐이더를 수행하기 전에 기하 모델을 구성하는 매 꼭지점마다 설정해주어야 하는 정보를 저장하고 있는 변수들을 모두 기술하라.

(f) 이 코드에서 버텍스 쉐이더를 수행하기 전에 기하 모델을 구성하는 매 꼭지점마다 설정할 필요가 없이 필요할 때만 설정해주면 되는 정보를 저장하고 있는 변수들을 모두 기술하라.

(g) 만약 첫 문장 바로 직후에 다음과 같은 문장을 삽입하면,

```
gl_Position.y *= -1.0f;
```

렌러링 결과가 어떻게 바뀔지 기술하라.

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

(a) (a)의 공식에서 Lambert의 코사인 법칙을 표현해주는 부분만 정확히 기술하라.

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

(c) (a)의 공식에서 카메라에서 바라보는 방향에 직접적으로 영향을 받는 변수를 모두 나열하라.

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

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

(a) 퐁의 조명 모델 식의 두 예 ($||N|| = ||V|| = ||L_i|| = ||R_i|| = ||H_i|| = 1$)

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

(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

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

(d) 스폿 광원 효과의 계산

Figure 7: 조명 모델 공식 예

(d) (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.

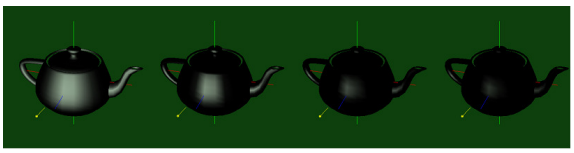
(e) (b)의 공식에서 (a)의 공식의 d_i에 해당하는 변수(또는 수식)를 기술하라.

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

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

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

(i) (b)의 공식에서 아래 그림이 암시하는 효과를 조절할 수 있는 변수를 기술하라.



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

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

8. 9쪽에는 OpenGL Shading Language를 사용하여 가급적 그림 7(b)의 퐁의 조명 모델 식에 기반을 두어 퐁 쉐이딩 방법을 구현한 버텍스 쉐이더와 프래그먼트 쉐이더 프로그램이 주어져 있다.

(a) 이 두 쉐이더 코드에서 OpenGL의 절단 좌표계 (Clip Coordinate) 공간에서의 값을 가지는 변수들을 모두 기술하라.

(b) 이 프래그먼트 쉐이더에서 `i`번째 광원 `u_light[i]`가 평행 광원 (parallel or direc-

tional light)이라면, 문맥 상 이 구조변수의 어떤 필드값이 어떤 값을 가져야 할까?

(c) 73번 문장 수행 후 `tmp_vec4.y`에 저장이 되는 값의 정확한 기하적인 의미를 기술하라.

(d) 점 광원을 사용할 경우 그림 7(a)의 조명 공식 (1)에서의 벡터 L_i 값이 계산이 되는 문장 번호를 기술하라.

(e) 그림 7(c)의 i번째 광원에 해당하는 k_{2i} 값을 저장하고 있는 변수 (또는 식)를 이 코드에서 찾아 OpenGL Shading Language 문법에 맞게 정확히 기술하라.

(f) 104번 문장에서 `-normalize(P_EC)`는 정확히 어느 지점에서 어느 지점을 향한 벡터인지 기술하라.

(g) 104번 문장에서 계산해주는 벡터 `H_EC`의 이름을 기술하라.

(h) 104번 문장의 (가) 부분에 들어갈 수식을 OpenGL Shading Language 문법에 맞게 정확히 기술하라.

(i) 105번 문장의 (나)와 (다)에 들어갈 두 벡터에 해당하는 기호 각각을 그림 7(a)의 조명 공식 (2)에서 찾아 기술하라. (참고: 순서는 상관 없음)

(j) 111번 문장의 (라)에 들어갈 변수의 값을 그림 7(b)의 조명 공식의 기호들을 사용하여 기술하라.

(k) 84번 문장의 (마) 부분에 들어갈 수식을 OpenGL Shading Language 문법에 맞게 정확히 기술하라.

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

9. 10쪽의 코드는 계층적 모델링 변환을 통하여 자동차를 세상 좌표계로 그려주는 OpenGL 프로그램이다. 이 코드와 그림 8을 보면서 답하라.

(a) 이 프로그램은 그림 8에 주어진 자동차를 구성하는 트리 구조를 어떤한 방식으로 탐색 하고 있는지 자료 구조 시간에 배운 컴퓨터공학 용어를 사용하여 기술하라.

(b) 프로그램 문맥상 62번 문장의 (A)에 들어갈 내용을 이 프로그램의 문법에 맞게 정확히 기술하라.

(c) 프로그램 문맥상 62번 문장의 (B), (C), 그리고 (D)에 들어갈 내용을 이 프로그램의 문법에 맞게 정확히 기술하라. (힌트: 그림 8의 내용을 잘 살펴볼 것)

(d) 이 프로그램의 26번 문장은 자동차가 움직임에 따라 두 앞 바퀴가 좌우로 회전하는 모습을 생성해주는 기하 변환을 계산하고 있다 (`draw_car()` 함수에서 적용이 됨). 문맥상 (E), (F), 그리고 (G)에 들어갈 내용을 이 프로그램의 문법에 맞게 정확히 기술하라.

(e) 이 프로그램의 29번 문장은 자동차가 움직임에 따라 네 바퀴가 진행 방향으로 회전하는 모습을 생성해주는 기하 변환을 계산하고 있다 (`draw_car()` 함수에서 적용이 됨). 문맥상 (H), (I), 그리고 (J)에 들어갈 내용을 이 프로그램의 문법에 맞게 정확히 기술하라.

(f) 이 프로그램에서 94번 문장 수행 후 변수 `ModelMatrix_CAR_WHEEL`에 저장되는 4행 4열의 내용을 그림 8의 행렬 기호를 사용하여 표현하라.

(g) 57번 문장의 `draw_wheel_and_nut()` 함수가 92번 문장에서 호출되어 수행이 되는 과정에서, 변수 `i`가 2일 때 63번 문장 수행 결과 변수 `ModelMatrix_CAR_NUT`에 저장되는 행렬의 내용을 그림 8의 행렬 기호를 사용하여 표현하라. 이 순간에 행렬 `car_status.Matrix_CAR_WHEEL_ROTATE`의 내용은 M_R이라고 가정함.

(h) 이 프로그램의 85번 문장은 자동차의 운전석에 배치한 카메라의 위치와 방향을 기술해주는 프레임을 그려주는 역할을 한다 (x, y, 그리고 z축 각각이 u, v, 그리고 n축에 대응함을 상기할 것). 이 자동차의 운전석에서 세상을 바라볼 때 오른쪽 방향은 자동차 바디 (`CAR_BODY`)의 모델링 좌표계를 기준으로 어느 축 방향을 향하고 있을까? "양의 x축" 또는 "음의 x축"과 같이 해당 축과 방향을 정확히 기술할 것.

(i) 이 프로그램의 92번 문장과 97번 문장이 그려주는 0번과 1번 바퀴와는 달리 104번 문장과 110번 문장이 그려주는 2번과 3번 바퀴에 대해서는 크기변환 (scale)이 수행되는 이유는 무엇일까?

(j) 이 프로그램의 42번 문장은 카메라를 운전석에 배치할 때의 뷰잉 변환 행렬을 계산해주고 있다. 이 문장에서 변수 `ModelMatrix_CAR_BODY_to_DRIVER`는 자동차 바디의 모델링 좌표계의 원점에 정렬되어 있는 카메라 프레임을 이 좌표계 상에서 운전석의 위치로 변환 시켜주는 강체변환을 저장하고 있다. 문맥 상 이 프로그램이

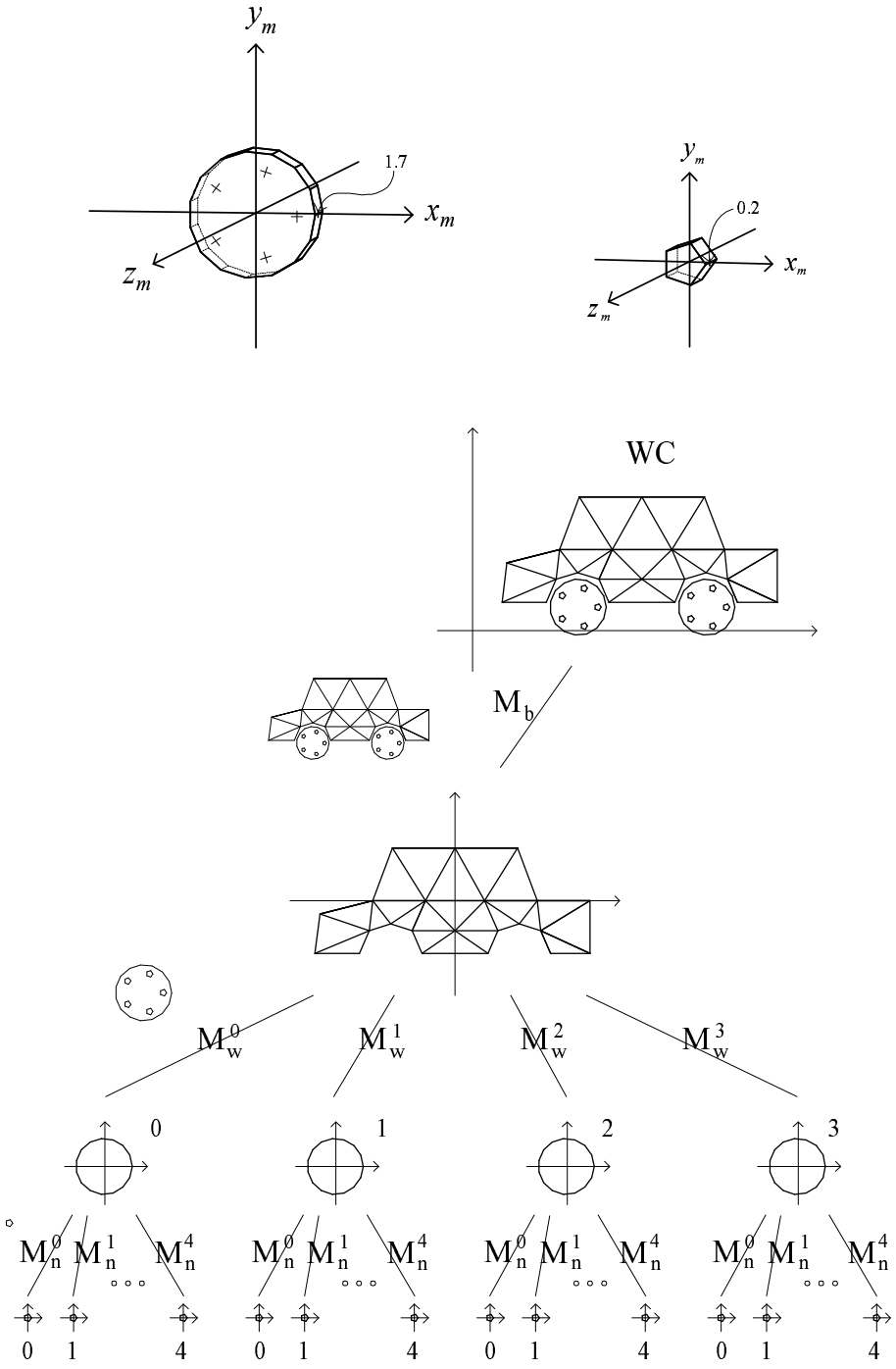


Figure 8: 자동차의 계층적 표현

제대로 작성하기 위하여 (K)에 들어갈 내용을 이 프로그램에서 사용한 변수를 사용하여 정확히 기술하라. (힌트: 뷰잉 변환은 세상 좌표에 존재하는 카메라 프레임을 좌표계 원점을 중심으로 정렬시켜주는 변환임을 상기할 것)

```
/**************************   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;
out vec3 v_position_EC;
out vec3 v_normal_EC;

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

    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; // compute this effect only if .w != 0.0f
    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;

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

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

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

    color_sum = u_material.emissive_color + u_global_ambient_color * u_material.ambient_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( (마) , spot_dir);
                if (tmp_float >= cos(radians(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(u_light[i].position.xyz);
        }

        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*u_material.diffuse_color*tmp_float;

                vec3 H_EC = normalize( (가) - normalize(P_EC));
                tmp_float = dot( (나) , (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) {
    final_color = lighting_equation(v_position_EC, normalize(v_normal_EC));
}
```

```
#define WHEEL_NUT_RADIUS               (1.7f - 0.5f)
#define WHEEL_NUT_Z_OFFSET             1.0f

#define FRONT_WHEEL_ROLL_ANGLE_DELTA   5.0f
#define FRONT_WHEEL_TURN_ANGLE_DELTA   2.5f
#define FRONT_WHEEL_TURN_ANGLE_MAX     45.0f
#define BODY_YAW_SENSITIVITY           0.1f

struct _CAR_STATUS {
    float body_yaw_angle;
    float front_wheel_roll_angle;
    float front_wheel_turn_angle;
    glm::mat4 Matrix_CAR_WHEEL_ROTATE;
    int flag_body_yaw_mode;
} car_status;

void update_car_body_transformation(void) {
    ModelMatrix_CAR_BODY = glm::rotate(glm::mat4(1.0f),
        TO_RADIAN*car_status.body_yaw_angle, glm::vec3(0.0f, 1.0f, 0.0f));

    ModelMatrix_CAR_BODY_INVERSE = glm::rotate(glm::mat4(1.0f),
        -TO_RADIAN*car_status.body_yaw_angle, glm::vec3(0.0f, 1.0f, 0.0f));
}

void update_car_front_wheel_rotate_matrix(void) {
    car_status.Matrix_CAR_WHEEL_ROTATE = glm::rotate(glm::mat4(1.0f),
        TO_RADIAN*car_status.front_wheel_turn_angle, glm::vec3( (E), (F), (G) ));

    car_status.Matrix_CAR_WHEEL_ROTATE = glm::rotate(car_status.Matrix_CAR_WHEEL_ROTATE,
        TO_RADIAN*car_status.front_wheel_roll_angle, glm::vec3( (H), (I), (J) ));
}

void set_ViewMatrix_for_world_viewer(void) {
    ViewMatrix = glm::mat4(camera_wv.uaxis.x, camera_wv.vaxis.x, camera_wv.naxis.x, 0.0f,
        camera_wv.uaxis.y, camera_wv.vaxis.y, camera_wv.naxis.y, 0.0f,
        camera_wv.uaxis.z, camera_wv.vaxis.z, camera_wv.naxis.z, 0.0f,
        0.0f, 0.0f, 0.0f, 1.0f);
    ViewMatrix = glm::translate(ViewMatrix, -camera_wv.pos);
}

void set_ViewMatrix_for_driver(void) {
    ViewMatrix = glm::affineInverse( (K) * ModelMatrix_CAR_BODY_to_DRIVER);
}

void initialize_car(void) {
    car_status.body_yaw_angle = 0.0f;
    update_car_body_transformation();
    if (camera_type == CAMERA_DRIVER) {
        set_ViewMatrix_for_driver();
        ViewProjectionMatrix = ProjectionMatrix * ViewMatrix;
    }
    car_status.front_wheel_roll_angle = 0.0f;
    car_status.front_wheel_turn_angle = 0.0f;
    update_car_front_wheel_rotate_matrix();
}

void draw_wheel_and_nut() {
    glUniform3f(loc_primitive_color, 0.000f, 0.808f, 0.820f); // color name: DarkTurquoise
    draw_geom_obj(GEOM_OBJ_ID_CAR_WHEEL); // draw wheel

    for (int i = 0; i < 5; i++) {
        ModelMatrix_CAR_NUT = glm::rotate( (A), TO_RADIAN*72.0f*i, glm::vec3( (B), (C), (D) ));
        ModelMatrix_CAR_NUT = glm::translate(ModelMatrix_CAR_NUT, glm::vec3(WHEEL_NUT_RADIUS, 0.
    0f, WHEEL_NUT_Z_OFFSET));
        ModelViewProjectionMatrix = ViewProjectionMatrix * ModelMatrix_CAR_NUT;
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix
    [0][0]);

        glUniform3f(loc_primitive_color, 0.690f, 0.769f, 0.871f); // color name: LightSteelBlue
        draw_geom_obj(GEOM_OBJ_ID_CAR_NUT); // draw i-th nut
    }
}

void draw_car(void) {
    glUniform3f(loc_primitive_color, 0.498f, 1.000f, 0.831f); // color name: Aquamarine
    draw_geom_obj(GEOM_OBJ_ID_CAR_BODY); // draw body

    glLineWidth(5.0f);
    draw_axes(); // draw MC axes of body
    glLineWidth(1.0f);

    ModelMatrix_CAR_DRIVER = glm::translate(ModelMatrix_CAR_BODY, glm::vec3(-3.0f, 0.5f, 2.5f));
    ModelMatrix_CAR_DRIVER = glm::rotate(ModelMatrix_CAR_DRIVER, TO_RADIAN*90.0f, glm::vec3(0.0f,
     1.0f, 0.0f));
    ModelViewProjectionMatrix = ViewProjectionMatrix * ModelMatrix_CAR_DRIVER;
    glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0]
    [0]);
    glLineWidth(5.0f);
    draw_axes(); // draw camera frame at driver seat
    glLineWidth(1.0f);

    ModelMatrix_CAR_WHEEL = glm::translate(ModelMatrix_CAR_BODY, glm::vec3(-3.9f, -3.5f, 4.5f));
    ModelMatrix_CAR_WHEEL *= car_status.Matrix_CAR_WHEEL_ROTATE;
    ModelViewProjectionMatrix = ViewProjectionMatrix * ModelMatrix_CAR_WHEEL;
    glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0]
    [0]);
    draw_wheel_and_nut();  // draw wheel 0

    ModelMatrix_CAR_WHEEL = glm::translate(ModelMatrix_CAR_BODY, glm::vec3(3.9f, -3.5f, 4.5f));
    ModelViewProjectionMatrix = ViewProjectionMatrix * ModelMatrix_CAR_WHEEL;
    glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0]
    [0]);
    draw_wheel_and_nut();  // draw wheel 1

    ModelMatrix_CAR_WHEEL = glm::translate(ModelMatrix_CAR_BODY, glm::vec3(-3.9f, -3.5f, -4.5f));
    ModelMatrix_CAR_WHEEL *= car_status.Matrix_CAR_WHEEL_ROTATE;
    ModelMatrix_CAR_WHEEL = glm::scale(ModelMatrix_CAR_WHEEL, glm::vec3(1.0f, 1.0f, -1.0f));
    ModelViewProjectionMatrix = ViewProjectionMatrix * ModelMatrix_CAR_WHEEL;
    glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0]
    [0]);
    draw_wheel_and_nut();  // draw wheel 2

    ModelMatrix_CAR_WHEEL = glm::translate(ModelMatrix_CAR_BODY, glm::vec3(3.9f, -3.5f, -4.5f));
    ModelMatrix_CAR_WHEEL = glm::scale(ModelMatrix_CAR_WHEEL, glm::vec3(1.0f, 1.0f, -1.0f));
    ModelViewProjectionMatrix = ViewProjectionMatrix * ModelMatrix_CAR_WHEEL;
    glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0]
    [0]);
    draw_wheel_and_nut();  // draw wheel 3
}
```