

기초 컴퓨터 그래픽스

HW3 README

20191559 강상원

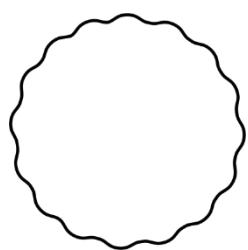
1. [환경 명세]

- Windows 11 64bit, Intel® Core™ i7-1255U CPU, Intel® Iris® Xe Graphics, Visual Studio Community 2022 Release x64

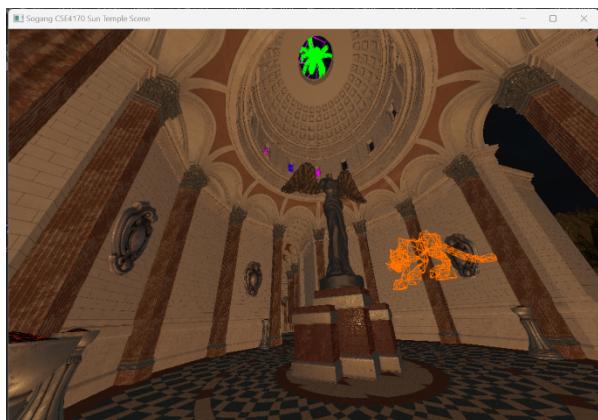
2. [요구사항]

1. 물체의 배치 및 움직임 구현

(a) 본 과목에서 제공하는 예제 프로그램 (5.1.Tiger Wireframe GLSL)을 활용하여, 움직이는 호랑이 (파일 12개로 구성)을 적절한 모델링 변환을 적용하여 가상의 세상에 돌아다니도록 하라.



호랑이는 기본적으로 왼쪽 모양과 같은 물결 모양 원형을 따라 움직이며, 바라보는 방향 또한 호랑이의 순간 이동 방향에 맞추어져 있다. 호랑이가 똑바로 움직이지 않고 주변을 살피면서 어슬렁 어슬렁 걸어가는 효과를 구현하였다.



또한 'p' camera에서 볼 수 있듯, 돔형 천장에 뚫린 구멍에 주기적으로 초록색 거미가 나타날 때마다 깜짝 놀라서 점프하는 동작을 취하고 있다. 점프를 할 때 단순히 선형적으로 위 아래로 움직이는 것이 아니라 탄성적으로 움직인다.

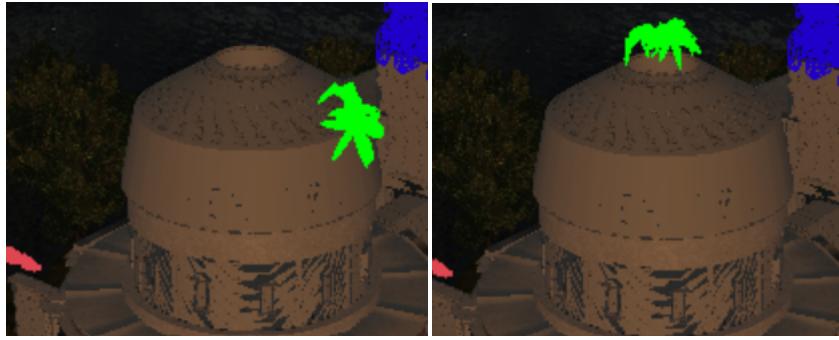
(b) 호랑이 물체는 키보드 또는 마우스 동작을 통하여 움직임과 멈춤을 조절할 수가 있어야 한다.

키보드의 'm'키를 누르면 움직이던 호랑이가 멈추거나, 이미 멈춘 상태라면 움직임을 재개한다.

(c) 본 과목에서 제공하는 또 다른 예제 프로그램 (4.0.1.3D Objects Anim 21)을 활용하여, 3D 기하 물체 중 동적인 물체 2개를 선택하여 가상의 세상에서 돌아다니도록 해라.

1. Spider

'u' camera나 'o' camera에서 볼 수 있듯이, 초록색 거미가 돔형 천장 위를 기어가듯 움직이고, 상단 구멍 부분에 잠시 머무는 듯한 효과를 연출하였다.



반구 형태의 천장을 따라 표면을 매끄럽게 움직이고, 거미의 배면이 향하는 방향도 반구의 중심 방향으로 되어 있다.

```
glm::vec3 center = glm::vec3(0.0f, 0.0f, 1700.0f);
float radius = 750.0f;
float theta = ((sin((timestamp_scene+80) / 30.0f) + 1) * glm::pi<float>());
// Range [0, 2pi]
float phi = ((cos((timestamp_scene+80) / 30.0f) + 1) * glm::pi<float>()) /
4.0f; // Range [0, pi/2]

glm::vec3 new_position;
new_position.x = center.x + radius * sin(phi) * cos(theta);
new_position.y = center.y + radius * sin(phi) * sin(theta);
new_position.z = center.z + radius * cos(phi);
spider_position = new_position;

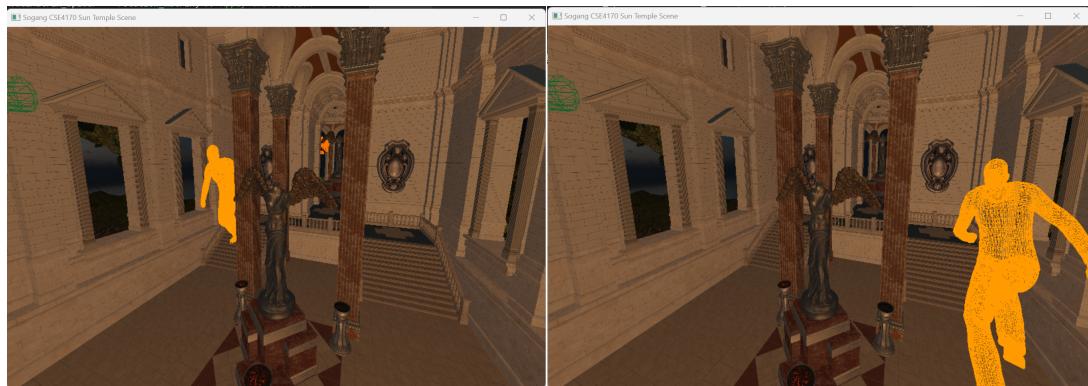
// Compute the direction from the spider's position to the center of the
hemisphere
glm::vec3 down_direction = glm::normalize(center - spider_position);

// Compute the rotation matrix that aligns the y-axis with the computed
direction
glm::vec3 y_axis = glm::vec3(0.0f, 1.0f, 0.0f); // Assuming the spider's
"down" direction is the negative y-axis
glm::vec3 axis = glm::cross(y_axis, down_direction);
float angle = glm::acos(glm::dot(y_axis, down_direction));
glm::mat4 rotation_matrix = glm::rotate(glm::mat4(1.0f), angle, axis);
```

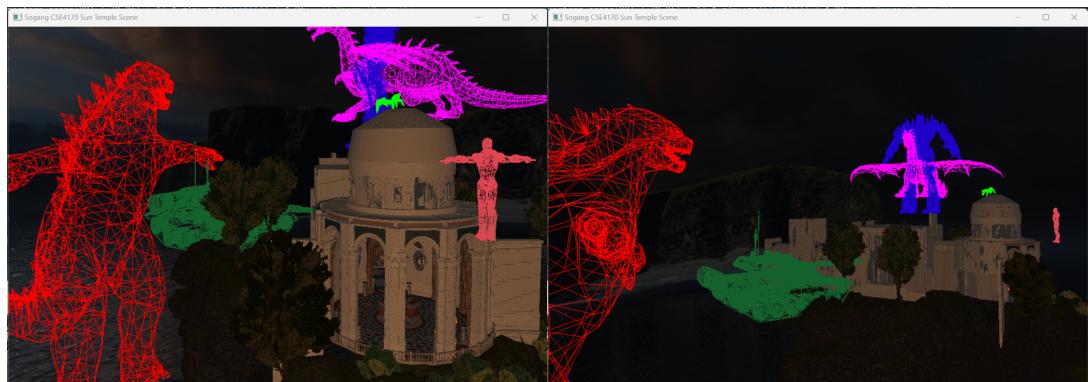
위와 같이 움직임을 구현하였다.

2. Ben

'i' camera에서 관찰할 수 있듯이, Ben 모델은 호랑이를 보고 놀라 혼비백산 사원의 내부를 달리고 있다. 이 때, 단순히 사각형으로 움직이는 것이 아니라 계단의 높낮이에 따라 실제 계단을 오르내리듯이 높낮이가 달라지며, 움직이는 방향에 따라 바라보는 방향이 달라진다. 또한 현실 세계처럼 계단을 오르거나 내릴 때는 평지를 달릴 때보다 속도가 느려진다. 다급함을 나타내기 위해서 중간중간 점프하면서 뛴박질을 한다. (계단을 오르면서 점프하거나, 평지에서 점프)



(d) 위의 예제 프로그램을 활용하여, 3D 물체 중 정적인 물체를 선택하여 최대 5개까지 적절한 모델링 변환을 통하여 세상에 배치하라.



Godzilla, tank, optimus, dragon, ironman 총 5개의 정적인 물체가 배치되어 있다.

각 모델의 크기는 실제 영화/매체 상에서 묘사된 것과 비슷한 비율로 나타나었으며, 바다에서 나타나 섬에 있는 사원을 공격하려는 고질라와 탱크, 그리고 사원 꼭대기에서 용을 타고 있는 옵티머스 프라임과 아이언맨을 관찰할 수 있다.

2. 가상 카메라의 배치 및 조절 기능 구현: 키보드의 'u', 'i', 'o', 'p', 'a', 't' 또는 'g' 키를 누르면 다음에 기술하는 방식의 카메라 모드에서 세상을 바라보도록 하라.

(a) 세상 관찰 카메라

i. 가상의 세상을 잘 관찰 할 수 있는 위치와 방향이 고정된 가상 카메라를 4대 배치하라: Camera u, Camera i, Camera o, Camera p.

각 카메라는 키보드의 'u', 'i', 'o', 또는 'p' 키로 선택할 수 있는데, 키를 누를 때마다 해당 카메라에서 바라본 세상이 윈도우 화면에 도시되어야 한다.



- Camera u:

전반적으로 멀리서 사원과 그 주변을 관찰하는 카메라이다.

고질라, 용, 옵티머스, 아이언맨, 탱크, 거미, 호랑이, 그리고 간간이 벽 너머 사람을 관측 가능하다.

- Camera i:

Ben 모델의 달리기를 잘 관측할 수 있는 사원 내부 카메라이다.

- Camera o:

사원의 모습을 공중에서 관측한 듯한 카메라이다.

용, 옵티머스, 거미, 아이언맨, 호랑이, 탱크를 관측 가능하다.

- Camra p:

거미와 호랑이의 움직임을 잘 관측할 수 있는, 시야 넓은 카메라이다.

거미가 나타날 때마다 호랑이가 놀라는 모습을 가장 관측하기 좋다.

ii. 각 세상 관찰 카메라 모드에서 CTRL 키를 누른 상태에서 마우스의 스크롤 훈을 사용하여 줌-인/줌-아웃이 되도록 하라. 이때 카메라의 위치와 방향을 고정한 상태에서 스크롤 방향에 따라 적절히 줌-인과 줌-아웃이 되어야 한다.

키보드 Ctrl 키를 누른 상태에서 마우스의 훈을 움직이면 카메라의 위치와 방향이 고정되어 있는 상태에서, 카메라의 fovy 값이 바뀌어 줌-인, 줌-아웃이 된다. u, i, o, p 각 카메라 뷰에서 작동한다.

```
void mousewheel_20191559(int wheel, int direction, int x, int y) {
    int key = glutGetModifiers();
    if (key == GLUT_ACTIVE_CTRL) {
        if (direction > 0) {
            camera_info[current_camera_index].fovy += 0.1;
            fprintf(stdout, "fovy: %lf\n",
camera_info[current_camera_index].fovy);
            ProjectionMatrix =
glm::perspective(camera_info[current_camera_index].fovy,
current_camera.aspect_ratio, current_camera.near_c, current_camera.far_c);
// ViewProjectionMatrix = ProjectionMatrix *
ViewMatrix;
            // glutPostRedisplay();
            fprintf(stdout, "CTRL + Zoom In\n");
        }
        else if (direction < 0) {
            camera_info[current_camera_index].fovy -= 0.1;
            fprintf(stdout, "fovy: %lf\n",
camera_info[current_camera_index].fovy);
            ProjectionMatrix =
glm::perspective(camera_info[current_camera_index].fovy,
current_camera.aspect_ratio, current_camera.near_c, current_camera.far_c);
// ViewProjectionMatrix = ProjectionMatrix *
ViewMatrix;
            // glutPostRedisplay();
            fprintf(stdout, "CTRL + Zoom Out\n");
        }
    }
    glutPostRedisplay();
}
```

(b) 세상 이동 카메라

i. a' 키를 누르면 세상 카메라 모드에서 세상 이동 카메라 모드로 변환한다:
Camera a. 프로그램 시작 후 초기에는 적절한 위치에 (자신이 배치한 물체들을 잘 관찰 할 수 있도록) 배치를 하며, 이 모드에서는 세상 이동 카메라에서 보이는 내용이 원도우 화면에 도시되어야 한다.



'a' 키를 누르면 고질라, 탱크, 옵티머스, 용, 거미를 잘 관측할 수 있는, 고질라 측면에서 사원을 바라본 카메라가 도시된다.

ii. 세상 이동 카메라 모드에서 키보드와 마우스를 적절히 사용하여 카메라가 3차원 공간에서 자유롭게 이동 (translation)하도록 하라. 즉 카메라가 좌-우/상-하/전-후 이동을 할 수 있어야 하며, 직관적으로 사용하기 쉽게 키보드/마우스 조작 기능을 설계한 후, 어떠한 방식으로 기능을 조절하는지 명시하라.

키보드의 ' \uparrow ' 키를 누르면, 카메라가 바라보는 방향으로 앞으로 움직이고, ' \downarrow '는 뒤로 움직인다. ' \leftarrow ' 키는 카메라가 왼쪽으로 움직이게 하고, ' \rightarrow '는 오른쪽으로 움직이게 한다.

`move_camera_20191559()` 함수로 기능을 구현하였다. 세상 좌표계 기준 좌-우/전-후가 아니라, 카메라 눈 좌표계 기준으로 움직인다.

```
void move_camera_20191559(Camera* camera, float dx, float dy, float dz) {
    if (current_camera_index == CAMERA_a) {
        camera->pos[0] += dx * camera->uaxis[0] - dy * camera->naxis[0] + dz * camera->vaxis[0];
        camera->pos[1] += dx * camera->uaxis[1] - dy * camera->naxis[1] + dz * camera->vaxis[1];
        camera->pos[2] += dx * camera->uaxis[2] - dy * camera->naxis[2] + dz * camera->vaxis[2];

        fprintf(stdout, "Camera moved to (%lf, %lf, %lf)\n",
                camera->pos[0], camera->pos[1], camera->pos[2]);
        // Update the view matrix
        set_ViewMatrix_from_camera_frame();
    }
}
```

Shift 키를 누른 상태에서 ' \uparrow ', ' \downarrow ' 키를 눌러 카메라를 상/하로 움직이게 한다.

`move_camera_vertical_20191559()` 함수로 기능을 구현하였다.

```
void move_camera_vertical_20191559(Camera* camera, float dy) {
    if (current_camera_index == CAMERA_a) {
        camera->pos[2] += dy;
        fprintf(stdout, "Camera moved to (%lf, %lf, %lf)\n",
                camera->pos[0], camera->pos[1], camera->pos[2]);
        // Update the view matrix
        set_ViewMatrix_from_camera_frame();
    }
}
```

iii. 세상 이동 카메라 모드에서 키보드와 마우스를 적절히 사용하여 카메라가 3 차원 공간에서 카메라 프레임 축, 즉 수업 시간에 배운 u, v, 그리고 n 축 둘레로 자유롭게 회전 (rotation)하도록 하라. 직관적으로 사용하기 쉽게 키보드/마우스 조작 기능을 설계한 후, 어떠한 방식으로 기능을 조절하는지 명시하라.

마우스의 왼쪽 버튼을 클릭한 상태에서, 좌-우로 드래그하면 카메라 n축 둘레로 회전하여 마치 카메라를 좌우로 회전하듯이 나타나고, 상-하로 드래그하면 v축 둘레로 회전한다. 마우스의 오른쪽 버튼을 누른 상태에서 위-아래로 드래그하면

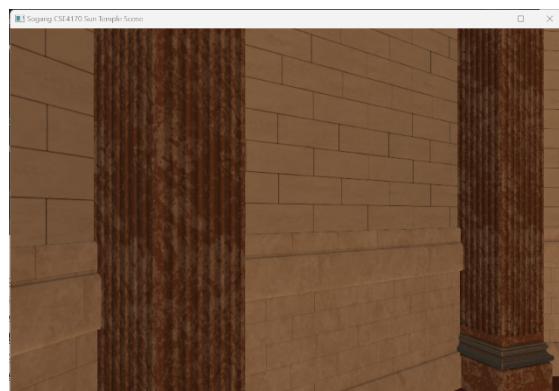
카메라가 n축 둘레로 회전한다. 3D 게임에서 마우스로 시선을 이동하는 방법과 같이 움직이며, 마우스를 누른 상태에서 대각선으로 움직여도 회전이 조합되어 움직여 조작이 편리하다.

iv. 세상 이동 카메라 모드에서 CTRL 키를 누른 상태에서 마우스의 스크롤 훨을 사용하여 줌-인/줌-아웃이 되도록 하라.

2-(a)-ii.와 마찬가지로, 키보드 Ctrl 키를 누른 상태에서 마우스의 훨을 움직이면 카메라의 fovy 값이 바뀌어 줌-인, 줌-아웃이 된다.

(c) 호랑이 관점 카메라

i. 't' 키를 누르면 1.(a) 항목에서 가상의 세상에 배치하여 돌아다니고 있는 호랑이의 눈에서 보이는 세상이 윈도우 화면에 도시되어야 한다: Camera t. 즉 카메라 프레임을 세상을 돌아다니고 있는 호랑이의 눈에 가상 카메라를 배치하여 세상을 바라보도록 하라.



호랑이가 움직임에 따라 호랑이의 right, up, direction vector가 달라지므로 이를 계산하여 ViewMatrix를 설정한다.
update_tiger_eye_camera() 함수에서 이를 설정한다.

```
void update_tiger_eye_camera_20191559() {  
    // Compute the world coordinates of the tiger's eye  
    glm::vec4 tiger_eye_model(0.0f, -88.0f, 62.0f, 1.0f);  
    glm::vec4 tiger_eye_world = ModelMatrix_tiger *  
        tiger_eye_model;  
  
    // Add a small nodding motion to the camera's y position  
    float nod_speed = 0.1f; // Adjust this value to control the  
    speed of the nodding motion  
    float nod_amplitude = 30.0f; // Adjust this value to control  
    the height of the nodding motion  
    tiger_eye_world.z += nod_amplitude * sin(timestamp_scene *  
        nod_speed);  
  
    // Compute the world coordinates of the tiger's direction  
    glm::vec4 tiger_direction_model(0.0f, -1.0f, 0.0f, 0.0f);  
    glm::vec4 tiger_direction_world = ModelMatrix_tiger *  
        tiger_direction_model;  
  
    // Normalize the direction  
    tiger_direction_world =  
        glm::normalize(tiger_direction_world);  
  
    // Create right vector (uaxis) by cross product of world's up
```

```

vector (Z axis) and direction
    glm::vec3 tiger_right_world = glm::cross(glm::vec3(0.0f,
0.0f, 1.0f), glm::vec3(tiger_direction_world));

        // Calculate up vector (vaxis) by cross product of direction
and right
        glm::vec3 tiger_up_world =
glm::cross(glm::vec3(tiger_direction_world), tiger_right_world);

        // Update the CAMERA_7 parameters in the camera_info array
Camera* pCamera = &camera_info[CAMERA_t];

        pCamera->pos[0] = tiger_eye_world.x; pCamera->pos[1] =
tiger_eye_world.y; pCamera->pos[2] = tiger_eye_world.z;
        pCamera->uaxis[0] = tiger_right_world.x; pCamera->uaxis[1] =
tiger_right_world.y; pCamera->uaxis[2] = tiger_right_world.z;
        pCamera->vaxis[0] = tiger_up_world.x; pCamera->vaxis[1] =
tiger_up_world.y; pCamera->vaxis[2] = tiger_up_world.z;
        pCamera->naxis[0] = -tiger_direction_world.x; pCamera-
>naxis[1] = -tiger_direction_world.y; pCamera->naxis[2] = -
tiger_direction_world.z;

        pCamera->fovy = 0.7f;

        if (current_camera_index == CAMERA_t)
            current_camera = *pCamera;

        set_ViewMatrix_from_camera_frame();
}

```

ii. 재미있는 효과를 생성하기 위하여 호랑이가 움직임에 따라 고개를 위-아래로
조금씩만 끄덕거리는 효과를 구현하라.

```

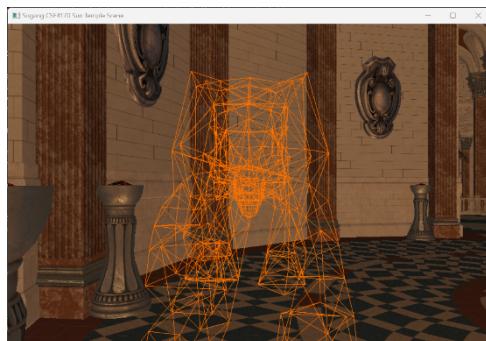
// Add a small nodding motion to the camera's y position
float nod_speed = 0.1f; // Adjust this value to control the
speed of the nodding motion
float nod_amplitude = 30.0f; // Adjust this value to control
the height of the nodding motion
tiger_eye_world.z += nod_amplitude * sin(timestamp_scene *
nod_speed);

```

nod_speed, nodamplitude로 호랑이의 고개 끄덕임 속도와 높이를 조절한다.

(d) 호랑이 관찰 카메라

i. 'g' 키를 누르면 움직이는 호랑이를 약간 뒤에서 쫓아가면서 호랑이를
관찰하는 카메라에서 보이는 세상이 윈도우 화면에 도시되어야 한다: Camera g.



`update_tiger_back_camera_20191559()` 함수로 호랑이 뒤에서 쫓아가는 카메라를 설정한다.

3. 추가 기능 구현: 위에서 기술한 요구 사항의 내용과 틀을 변경하지 않는 선에서 “재미있고 창의적인” 3D 그래픽스 효과를 생성하라.

거미의 움직임 로직은 다음과 같이 구현하여, 반구 위를 미끄러지듯 이동하다 꼭대기 (구멍 뚫린 부분)에서 잠시 머물다 가도록 구현하였다.

```
glm::vec3 center = glm::vec3(0.0f, 0.0f, 1700.0f);
float radius = 750.0f;
float theta = ((sin((timestamp_scene+80) / 30.0f) + 1) * glm::pi<float>()); // Range [0, 2pi]
float phi = ((cos((timestamp_scene+80) / 30.0f) + 1) * glm::pi<float>()) / 4.0f; // Range [0, pi/2]

glm::vec3 new_position;
new_position.x = center.x + radius * sin(phi) * cos(theta);
new_position.y = center.y + radius * sin(phi) * sin(theta);
new_position.z = center.z + radius * cos(phi);
spider_position = new_position;
```

$$\theta = \sin((t + 80) / 30) + 1) * \pi$$

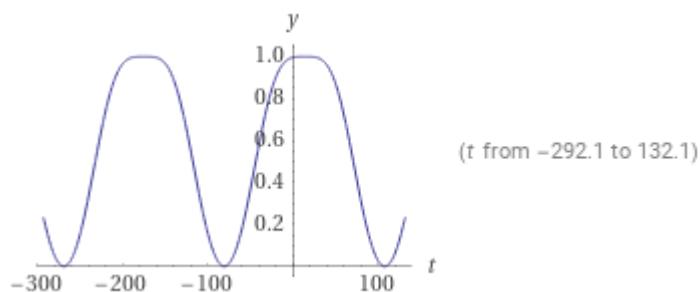
$$\varphi = (\cos((t + 80) / 30) + 1) * \pi / 4$$

$$\therefore 0 \leq \theta \leq 2\pi, 0 \leq \varphi \leq \pi/2$$

x좌표: $x + r * \sin(\varphi) * \cos(\theta)$

y좌표: $y + r * \sin(\varphi) * \sin(\theta)$

z좌표: $z + r * \cos(\varphi)$ 이므로 구형 위를 움직이면서 최고 좌표 지점 근방에서 오래 머문다.



$$y = \cos \left(\frac{\left(\cos \left(\frac{t + 80}{30} \right) + 1 \right) * \pi}{4} \right)$$

$\cos \left(\frac{\left(\cos \left(\frac{t + 80}{30} \right) + 1 \right) * \pi}{4} \right) = 1$ 이 될 때 (즉, 거미가 최고점에 도달했을 때의 주기를 계산하여 호랑이의 점프 주기를 맞추었다. (주기: 60π)

$$t = 10(6\pi n - 8 - 3 \cos^{-1}((4 - \pi)/\pi)), \quad n \in \mathbb{Z}$$

↓ draw_tiger_20191559() 의 일부

```
static const float jump_interval = 60.0f * glm::pi<float>();
static const float jump_duration = 20.0f; // Jump duration in seconds
static const float max_jump_height = 200.0f;

...

float time_since_last_jump = fmod(timestamp_scene - jump_start_time,
jump_interval);
if (time_since_last_jump < jump_duration) {
    // We are in a jump, calculate height
    float progress_through_jump = time_since_last_jump / jump_duration;
    float height = max_jump_height * (4 * progress_through_jump * (1 -
progress_through_jump));
    // tiger_position.z += height;
    ModelMatrix_tiger = glm::translate(ModelMatrix_tiger, glm::vec3(0, 0,
height));
}
```