2016-1 컴퓨터그래픽스 설계과제

20113337 최영근 / 20135188 정지은

1. 물체 변환&표현

1)Object::draw
2)earth
3) bumblebee & bumblebeeCar
4) Ironhide & IronhideCar
5) Mirage & MirageCar
6) Optimus & OptimusCar
7) Soundwave & SoundwaveCar

2. 네비게이션

1) 키보드
2) 충돌 감지

3. 배경

4. 마우스

1 물체 변환&표현(Object::draw)

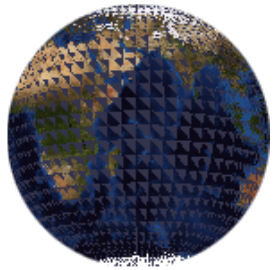
- 쉐이더변수에 각각 오브젝트의 버텍스정보를 전달함

```
void Object::draw(int loc_a_vertex, int loc_a_normal, int loc_a_texcoord)
{
  for (size_t i=0; i<m_groups.size(); ++i)
  {
    Group& group = m_groups[i];

     glVertexAttribPointer(loc_a_texcoord, 2, GL_FLOAT, false, 0, group.m_texcoords.data());
     glEnableVertexAttribArray(loc_a_texcoord);

     glVertexAttribPointer(loc_a_normal, 3, GL_FLOAT, false, 0, group.m_normals.data());
     glEnableVertexAttribArray(loc_a_normal);

     glVertexAttribPointer(loc_a_vertex, 3, GL_FLOAT, false, 0, group.m_vertices.data());
     glEnableVertexAttribArray(loc_a_vertex);
     glDrawArrays(GL_TRIANGLES, 0, group.m_vertices.size());

     glDisableVertexAttribArray(loc_a_normal);
     glDisableVertexAttribArray(loc_a_vertex);
     glDisableVertexAttribArray(loc_a_texcoord);
  }
}
```

1 물체 변환&표현(earth)

```
//draw earth
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex1_earth);
glUniform1i(loc_u_texid, 0);

mat_Model = glm::mat4(1.0);
mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-5.0, 10.0, 3.8));
mat_Model *= glm::rotate(earth_rotate_degree,glm::vec3(0,1,0));
mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

mat_PVM = mat_Proj*mat_View*mat_Model;
mat_MV = mat_View*mat_Model;
glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
g_earth.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);
```

T
R
S



▲부분캡쳐

R(R(Sp)) = TRSx

1 물체 변환&표현(bumblebee & bumblebeeCar)

```
    //draw BumbleBee
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex2_Bumblebee);
    glUniform1i(loc_u_texid, 0);

    mat_Model = glm::mat4(1.0);
T   mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(10.0, 0.0, 10.0));
R   mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
    mat_Model *= glm::rotate(PI/2,glm::vec3(0,0,-1));
    mat_Model *= glm::rotate(PI/4,glm::vec3(0,0,-1));
S   mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

    mat_PVM = mat_Proj*mat_View*mat_Model;
    mat_MV = mat_View*mat_Model;
    glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
    glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
    glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
    g_BumbleBee.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);

    //draw BumbleBeeCar
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex1_Bumblebee);
    glUniform1i(loc_u_texid, 0);

    mat_Model = glm::mat4(1.0);
T   mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(6.5, 0.0, 9.0));
R   mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
    mat_Model *= glm::rotate(PI/2,glm::vec3(0,0,-1));
    mat_Model *= glm::rotate(PI/4,glm::vec3(0,0,-1));
S   mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

    mat_PVM = mat_Proj*mat_View*mat_Model;
    mat_MV = mat_View*mat_Model;
    glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
    glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
    glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
    g_BumbleBee1.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);
```

▲부분캡쳐

R(R(Sp)) = TRSx

1 물체 변환&표현(Ironhide & IronhideCar)

```
  //draw Ironhide
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, tex1_IronHide);
  glUniform1i(loc_u_texid, 0);

  mat_Model = glm::mat4(1.0);
T mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-15.0, 0.0, 0.0));
R mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
  mat_Model *= glm::rotate(PI/6,glm::vec3(0,0,-1));
S mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

  mat_PVM = mat_Proj*mat_View*mat_Model;
  mat_MV = mat_View*mat_Model;
  glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
  glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
  glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
  g_IronHide.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);

  //draw IronHideCar
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, tex2_IronHide);
  glUniform1i(loc_u_texid, 0);

  mat_Model = glm::mat4(1.0);
T mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-12.0, 0.0, -3.0));
R mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
  mat_Model *= glm::rotate(-PI/4,glm::vec3(0,0,1));
S mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

  mat_PVM = mat_Proj*mat_View*mat_Model;
  mat_MV = mat_View*mat_Model;
  glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
  glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
  glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
  g_IronHide1.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);
```

▲부분캡쳐

R(R(Sp)) = TRSx

1 물체 변환&표현(Mirage & MirageCar)

```
  //draw Mirage
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, tex1_Mirage);
  glUniform1i(loc_u_texid, 0);

  mat_Model = glm::mat4(1.0);
T mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-19.0, 0.0, 8.0));
R mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
  mat_Model *= glm::rotate(PI/6,glm::vec3(0,0,-1));
S mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

  mat_PVM = mat_Proj*mat_View*mat_Model;
  mat_MV = mat_View*mat_Model;
  glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
  glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
  glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
  g_Mirage.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);

  //draw MirageCar
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, tex2_Mirage);
  glUniform1i(loc_u_texid, 0);

  mat_Model = glm::mat4(1.0);
T mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-17.0, 0.0, 6.0));
R mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
  mat_Model *= glm::rotate(PI/6,glm::vec3(0,0,-1));
S mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

  mat_PVM = mat_Proj*mat_View*mat_Model;
  mat_MV = mat_View*mat_Model;
  glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
  glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
  glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
  g_Mirage1.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);
```

▲부분캡쳐

R(R(Sp)) = TRSx

1 물체 변환&표현(Optimus & OptimusCar)

```
    //draw Optimus
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex1_Optimus);
    glUniform1i(loc_u_texid, 0);

    mat_Model = glm::mat4(1.0);
T   mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(2.0, 0.0, -3.5));
R   mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
    mat_Model *= glm::rotate(PI/2,glm::vec3(0,0,-1));
    mat_Model *= glm::rotate(PI/4,glm::vec3(0,0,-1));
S   mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

    mat_PVM = mat_Proj*mat_View*mat_Model;
    mat_MV = mat_View*mat_Model;
    glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
    glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
    glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
    g_Optimus.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);

    //draw OptimusCar
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex2_Optimus);
    glUniform1i(loc_u_texid, 0);

    mat_Model = glm::mat4(1.0);
T   mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-2.0, 0.0, -6.0));
R   mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
    mat_Model *= glm::rotate(PI/2,glm::vec3(0,0,-1));
    mat_Model *= glm::rotate(PI/4,glm::vec3(0,0,-1));
S   mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

    mat_PVM = mat_Proj*mat_View*mat_Model;
    mat_MV = mat_View*mat_Model;
    glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
    glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
    glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
    g_Optimus1.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);
```

▲부분캡쳐

R(R(Sp)) = TRSx

1 물체 변환&표현(Soundwave & SoundwaveCar)

```
//draw Soundwave
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex1_Soundwave);
glUniform1i(loc_u_texid, 0);

mat_Model = glm::mat4(1.0);
T mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(6.0, 0.0, 3.8));
R mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
  mat_Model *= glm::rotate(PI/2,glm::vec3(0,0,-1));
  mat_Model *= glm::rotate(PI/4,glm::vec3(0,0,-1));
S mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

mat_PVM = mat_Proj*mat_View*mat_Model;
mat_MV = mat_View*mat_Model;
glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
g_Soundwave.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);

//draw SoundwaveCar
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex2_Soundwave);
glUniform1i(loc_u_texid, 0);

mat_Model = glm::mat4(1.0);
T mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(2.0, 0.0, 2.0));
R mat_Model *= glm::rotate(-PI/2,glm::vec3(1,0,0));
  mat_Model *= glm::rotate(PI/2,glm::vec3(0,0,-1));
  mat_Model *= glm::rotate(PI/4,glm::vec3(0,0,-1));
S mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(0.01f));

mat_PVM = mat_Proj*mat_View*mat_Model;
mat_MV = mat_View*mat_Model;
glUniformMatrix4fv(loc_mv_matrix, 1, false, glm::value_ptr(mat_MV));
glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
glUniformMatrix3fv(loc_normal_matrix, 1, false, glm::value_ptr(glm::transpose(glm::inverse(glm::mat3(mat_MV)))));
g_Soundwave1.draw(loc_a_vertex, loc_a_normal, loc_a_texcoord);
```

▲부분캡쳐

R(R(Sp)) = TRSx

1 물체 변환&표현(vertex shader)

```
#version 120

uniform mat4 u_pvm_matrix;
uniform mat4 mv_matrix;
uniform mat3 normal_matrix;

attribute vec4 a_vertex;
attribute vec3 a_normal;
attribute vec2 a_texcoord;

varying vec2 v_texcoord;
varying vec3 v_vertex;
varying vec3 v_normal;

varying mat4 modelview_matrix;

void main() {
  modelview_matrix = mv_matrix;

  v_normal = normalize(normal_matrix * a_normal);
  v_vertex = (mv_matrix * a_vertex).xyz;
  v_texcoord = a_texcoord;

  gl_Position = u_pvm_matrix * a_vertex;
}
```

Fragment 쉐이더에서 택스쳐의 색과 phong 모델을 계산하기 위해 Varying 변수로 전달

각 오브젝트마다 계산한 mvp 메트릭스와 오브젝트의 정점을 가지고 물체의 포지션 설정

1 물체 변환&표현(fragment shader)

```
#version 120

uniform sampler2D u_texid;
uniform mat3 normal_matrix;

varying vec2 v_texcoord;
varying vec3 v_normal;
varying vec3 v_vertex;

varying mat4 modelview_matrix;

vec4 light_position = vec4(-5.0, 10.0, 3.8, 1.0);
vec4 light_ambient = vec4(0.3, 0.1, 0.1, 1.0);
vec4 light_diffuse = vec4(0.5, 0.5, 0.5, 1.0);
vec4 light_specular = vec4(1.0, 1.0, 1.0, 1.0);
vec4 material_ambient = vec4(0.1, 0.1, 0.1,1.0);
vec4 material_diffuse = vec4(0.3, 0.3, 0.2, 1.0);
vec4 material_specular = vec4(0.1, 0.1, 0.1, 1.0);
float material_shininess = 10.0;

vec4 phong_reflection(vec3 normal_v){

        vec4 color = vec4(0,0,0,1);
        vec3 lightpos = (modelview_matrix * light_position).xyz;
        vec3 normal = normalize(normal_matrix * normal_v);
        vec3 light_vector = normalize(lightpos - v_normal );
        vec3 reflect_vector = reflect(-light_vector, normal);
        vec3 view_vector = normalize(-v_vertex);

        float kd = max(dot(light_vector, normal), 0.0);
        float ks = pow(max(0.0, dot(reflect_vector, view_vector) ), material_shininess);

        vec4 ambient = material_ambient * light_ambient;
        vec4 diffuse = kd*material_diffuse;
        vec4 specular = ks * material_specular;

        color = ambient + diffuse + specular;

        return color;
}

void main() {
  vec3 normal = texture2D(u_texid, v_texcoord).xyz;
  gl_FragColor = texture2D(u_texid, v_texcoord) + phong_reflection(normal);
}
```

Phong reflective 모델 계산을 위해 빛의 정보 (포지션, ambient, diffuse, specular),
물체가 갖는 정보(ambient, diffuse, specular) 를 설정

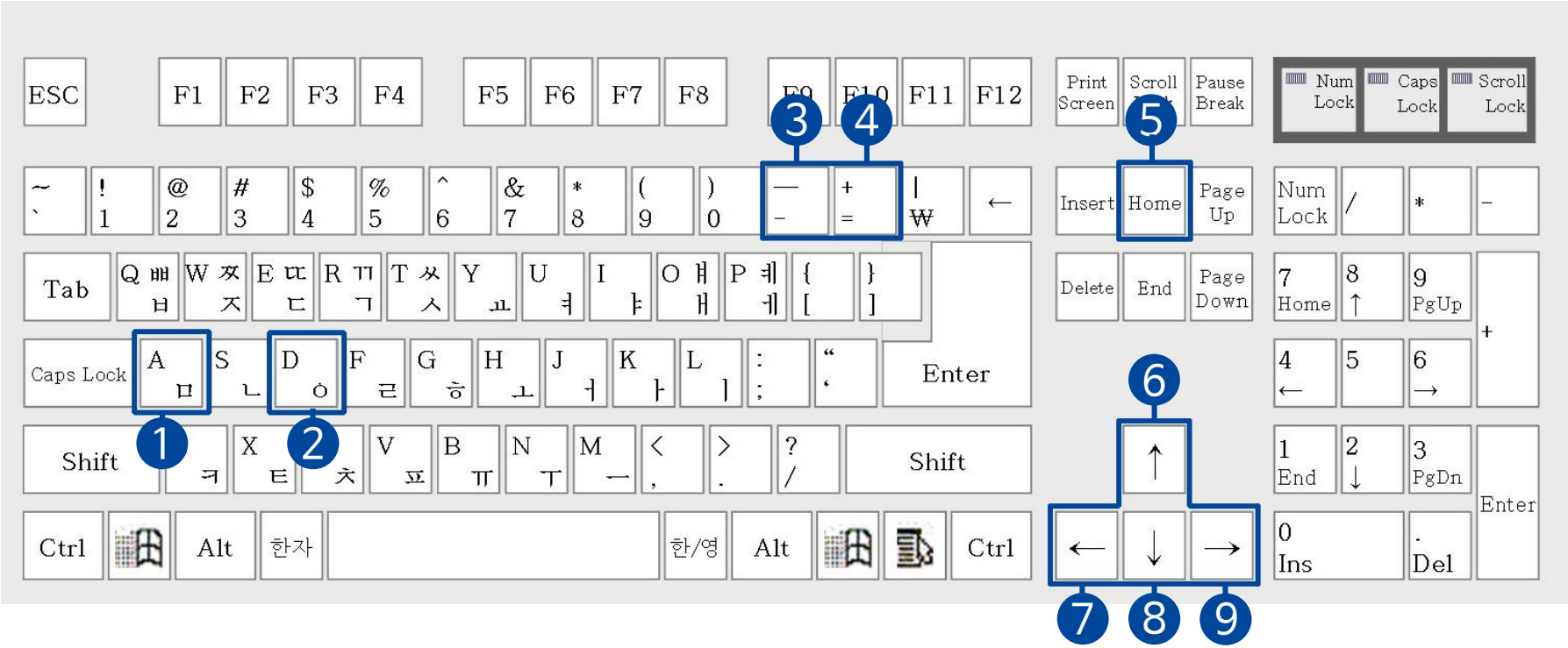
Phong reflective 모델 계산

L, N, R, V 값 계산

계산하여 얻은 ambient, diffuse, specular 값을
더해서 퐁 모델 계산

Phong 모델 적용

2.1 키보드(기능설명)



1. rotate_left
2. rotate_right
3. move_backward
4. move_forward
5. HOME
6. move_up
7. move_left
8. move_down
9. move_right

2.1 키보드(코드)

1 rotate_left

```
void Camera::rotate_left(float delta)
{
  front_dir_ = glm::rotateY(front_dir_, delta);
  right_dir_ = glm::cross(front_dir_, up_dir_);
}
```

2 rotate_right

```
void Camera::rotate_right(float delta)
{
  rotate_left(-delta);
}
```

3 move_backward

```
void Camera::move_backward(float delta)
{
  move_forward(-delta);
}
```

4 move_forward

```
void Camera::move_forward(float delta)
{
  position_ += delta * front_dir_;
}
```

5 HOME

```
void Camera::HOME()
{
  position_=glm::vec3(-5.0,3,40);
  front_dir_ = glm::vec3(0,0,-1);
  up_dir_=glm::vec3(0,1,0);
  right_dir_=glm::vec3(1,0,0);
  fovy_=45;
}
```

6 move_up

```
void Camera::move_up(float delta)
{
  position_+=delta*up_dir_;
}
```

7 move_left

```
void Camera::move_left(float delta)
{
  position_ -= delta * right_dir_;
}
```

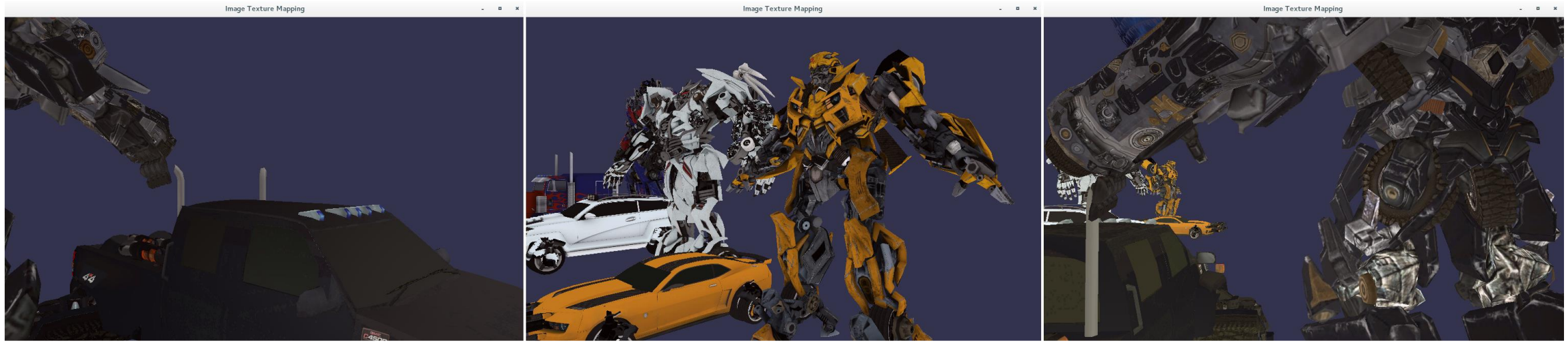
8 move_down

```
void  Camera::move_down(float delta)
{
  position_+=-delta*up_dir_;
}
```

9 move_right

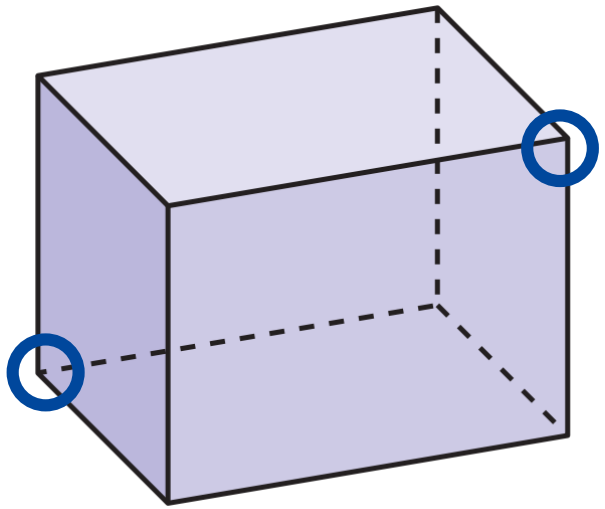
```
void Camera::move_right(float delta)
{
  move_left(-delta);
}
```

2.2 충돌감지



- 키보드를 이용하여 접근, 장면에 놓인 물체를 뚫고 지나가지 않도록 방지함

2.2 충돌감지(코드 및 표현방법 기술-1)



```
void create_min_max(Object& obj, glm::vec3 trans, float scale) {

    obj.max_x(trans.x, scale); obj.max_y(trans.y, scale); obj.max_z(trans.z, scale);
    obj.min_x(trans.x, scale); obj.min_y(trans.y, scale); obj.min_z(trans.z, scale);

    std::cout << obj.get_max_pos( ).x << " " << obj.get_max_pos( ).y << " " << obj.get_max_pos( ).z << std::endl;
    std::cout << obj.get_min_pos( ).x << " " << obj.get_min_pos( ).y << " " << obj.get_min_pos( ).z << std::endl;
}
```

Create_min_max : 최대, 최소의 점을 구해서 바운드박스를 생성

2.2 충돌감지(코드 및 표현방법 기술-2)

```
void keyboard(unsigned char key, int x, int y)
{
    if ('a' == key || 'A' == key) { ... }
    else if ('d' == key || 'D' == key) { ... }
    else if ('s' == key || 'S' == key) { ... }
    else if ('w' == key || 'W' == key) { ... }
    else if ('+' == key || '=' == key)  {
        if (check_foward_camera())
            g_camera.move_forward(1.0);
    }
    else if ('-' == key || '_' == key)  {
        if (check_backward_camera())
            g_camera.move_backward(1.0);
    }
    glutPostRedisplay();
}
void special(int key, int x, int y)
{
    switch (key)
    {
    case GLUT_KEY_UP:
        if (check_up_camera())
            g_camera.move_up(1.0);
        break;
    case GLUT_KEY_DOWN:
        if (check_down_camera())
            g_camera.move_down(1.0);
        break;
    case GLUT_KEY_LEFT:
        if (check_left_camera())
            g_camera.move_left(1.0);
        break;
    case GLUT_KEY_RIGHT:
        if (check_right_camera())
            g_camera.move_right(1.0);
        break;
    case GLUT_KEY_HOME:
        g_camera.HOME();
        break;
    default:
        break;
    }
    glutPostRedisplay();
}
```

```
bool  check_foward(Object& obj, Camera cam) {
    if (obj.get_min_x() - 1 <= cam.check_move_forward(1).x && cam.check_move_forward(1).x <= obj.get_max_x() + 1) {
        if (obj.get_min_y() - 1 <= cam.check_move_forward(1).y && cam.check_move_forward(1).y <= obj.get_max_y() + 1) {
            if (obj.get_min_z() - 1 <= cam.check_move_forward(1).z && cam.check_move_forward(1).z <= obj.get_max_z() + 1) {
                return false;
            }
        }
    }
    return true;
}
bool  check_backward(Object& obj, Camera cam) { ... }
bool  check_up(Object& obj, Camera cam) { ... }
bool  check_down(Object& obj, Camera cam) { ... }
bool  check_right(Object& obj, Camera cam) { ... }
bool  check_left(Object& obj, Camera cam) { ... }
bool check_foward_camera() {
    if (!check_foward(g_earth, g_camera))
        return false;
    if (!check_foward(g_BumbleBee, g_camera))
        return false;
    if (!check_foward(g_BumbleBee1, g_camera))
        return false;
    if (!check_foward(g_Soundwave, g_camera))
        return false;
    if (!check_foward(g_Soundwave1, g_camera))
        return false;
    if (!check_foward(g_Optimus, g_camera))
        return false;
    if (!check_foward(g_Optimus1, g_camera))
        return false;
    if (!check_foward(g_IronHide, g_camera))
        return false;
    if (!check_foward(g_IronHide1, g_camera))
        return false;
    if (!check_foward(g_Mirage, g_camera))
        return false;
    if (!check_foward(g_Mirage1, g_camera))
        return false;

    return true;
}
bool check_backward_camera() { ... }
bool check_up_camera() { ... }
bool check_down_camera() { ... }
bool check_right_camera() { ... }
bool check_left_camera() { ... }
```

- 예를 들어 +키를 눌렀을 때, check_forward_camera()가 실행되어 카메라의 포지션과 앞으로 갔을 때 각각의 오브젝트가 가진 바운딩박스와 충돌 계산

2.2 충돌감지(코드 및 표현방법 기술-3)

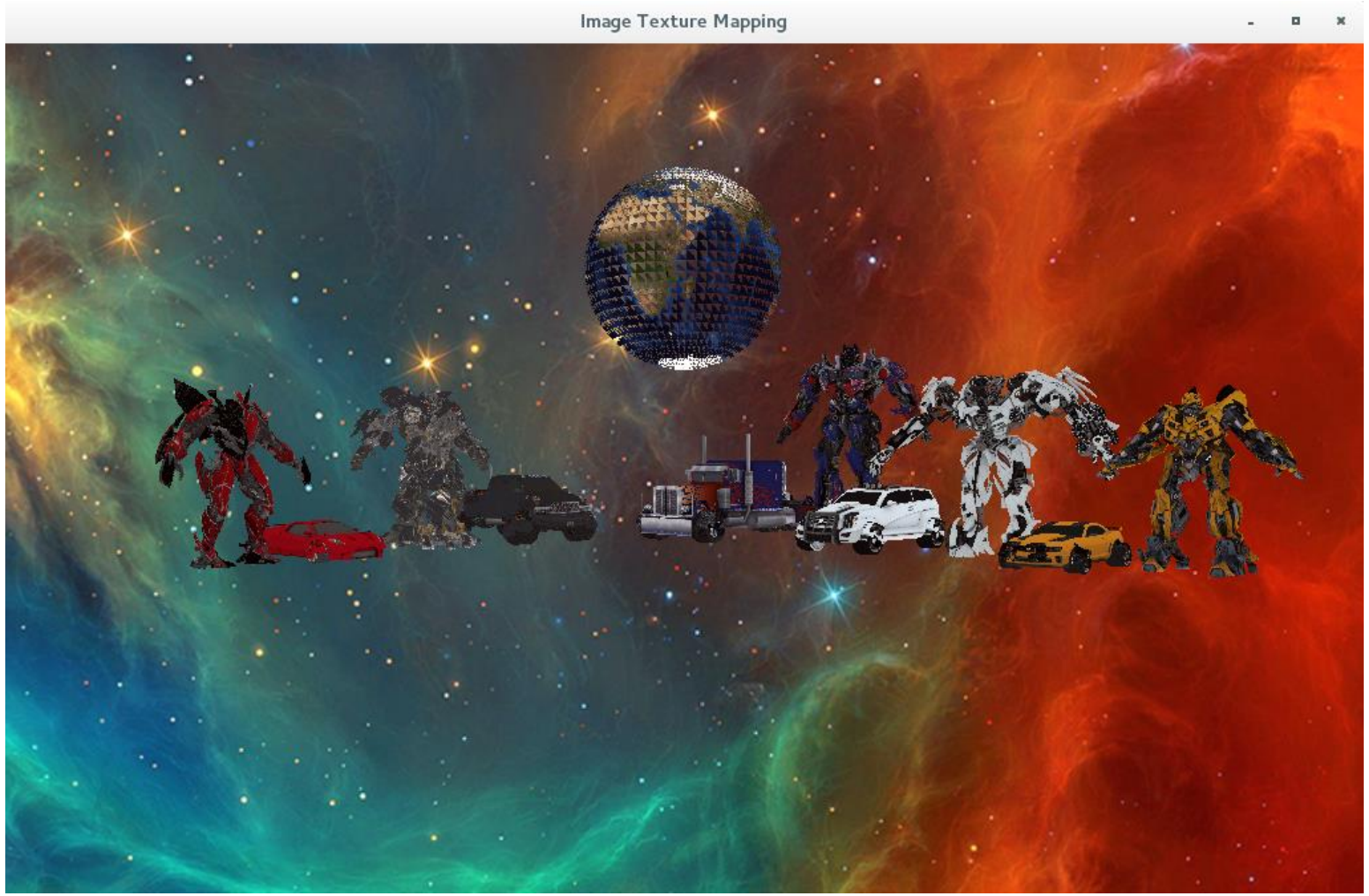
```c
void keyboard(unsigned char key, int x, int y)
{
    if ('a' == key || 'A' == key){ ... }
    else if ('d' == key || 'D' == key){ ... }
    else if ('s' == key || 'S' == key){ ... }
    else if ('w' == key || 'W' == key){ ... }
    else if ('+' == key || '=' == key)  {
        if (check_foward_camera())
            g_camera.move_forward(1.0);
    }
    else if ('-' == key || '_' == key)  {
        if (check_backward_camera())
            g_camera.move_backward(1.0);
    }
    glutPostRedisplay();
}
void special(int key, int x, int y)
{
    switch (key)
    {
    case GLUT_KEY_UP:
        if (check_up_camera())
            g_camera.move_up(1.0);
        break;
    case GLUT_KEY_DOWN:
        if (check_down_camera())
            g_camera.move_down(1.0);
        break;
    case GLUT_KEY_LEFT:
        if (check_left_camera())
            g_camera.move_left(1.0);
        break;
    case GLUT_KEY_RIGHT:
        if (check_right_camera())
            g_camera.move_right(1.0);
        break;
    case GLUT_KEY_HOME:
        g_camera.HOME();
        break;
    default:
        break;
    }
    glutPostRedisplay();
}
```

```c
glm::vec3 Camera::check_move_forward(float delta){
  glm::vec3 pos = position_;
  pos+= delta * front_dir_;
  return pos;
}
glm::vec3 Camera::check_move_backward(float delta){
  return check_move_forward(-delta);
}
glm::vec3 Camera::check_move_left(float delta){ ... }
glm::vec3 Camera::check_move_right(float delta){ ... }
glm::vec3 Camera::check_move_up(float delta){ ... }
glm::vec3 Camera::check_move_down(float delta){ ... }
```

- 만약 바운드박스와 충돌하지 않는다면, check_move_forward가 실행,
  새로운 pos 변수를 생성하여 delta값과 그 방향의 direction 변수를 곱하여 반환함

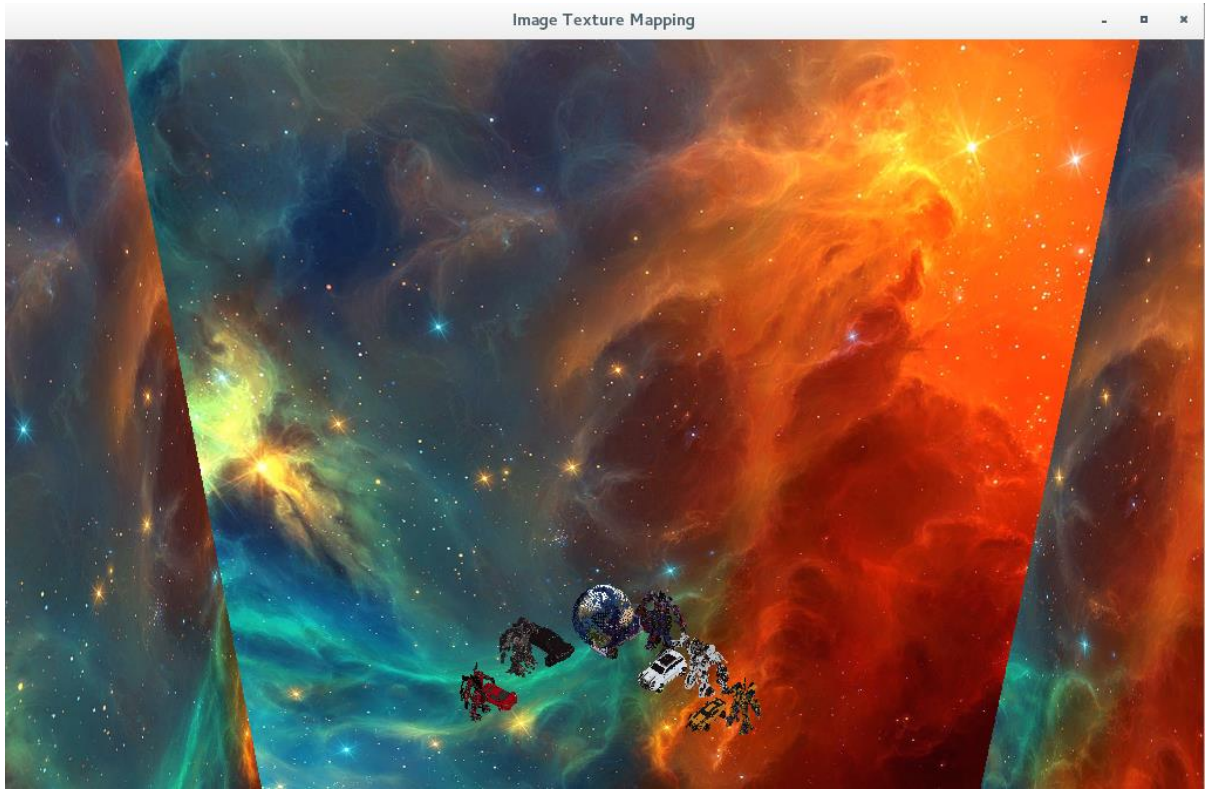
  (다른 방향도 모두 동일한 방법이므로 생략)

3 배경



〈처음 실행시켰을 때의 모습〉

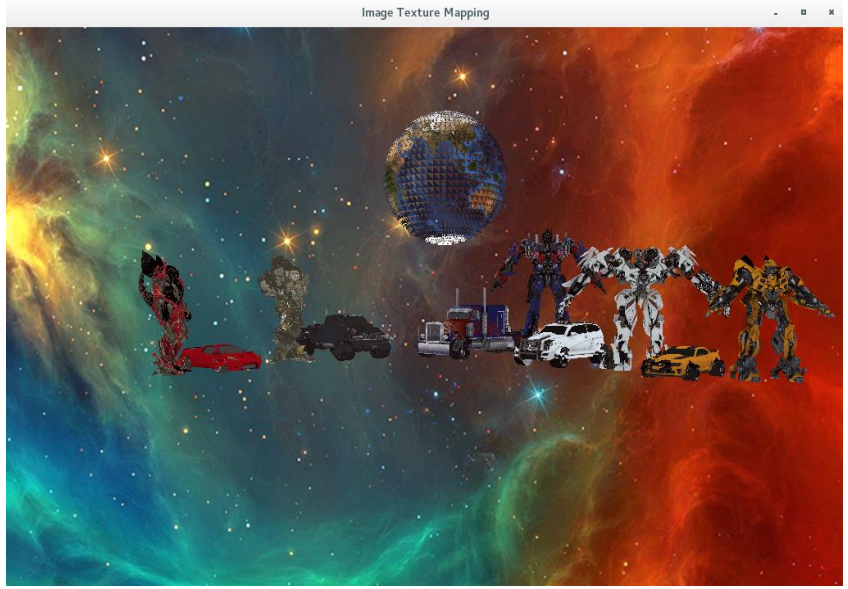
3 배경(code)

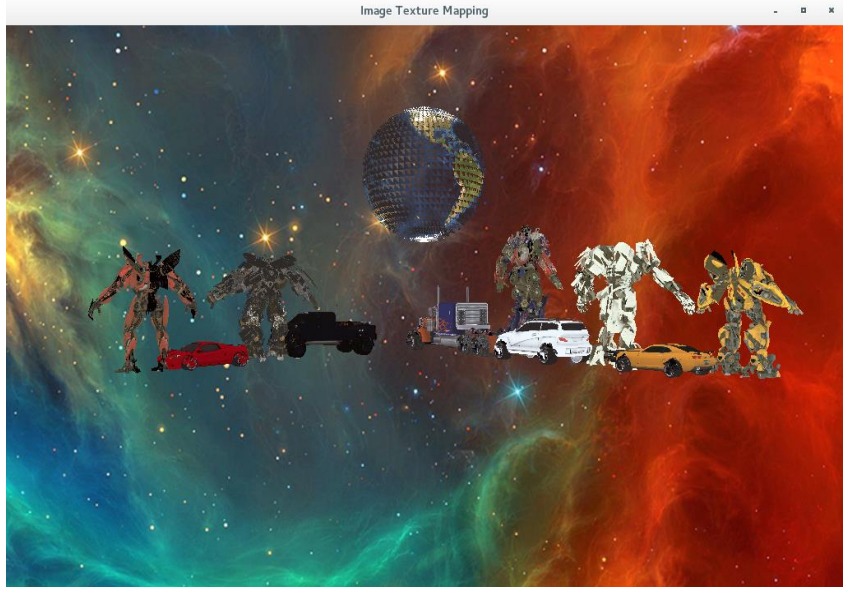



```
void background(){
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, textureid1);
  glUniform1i(loc_u_texid, 0);

    mat_Model = glm::mat4(1.0);
    mat_Model *= glm::translate(glm::mat4(1.0f), glm::vec3(-3.0, 14.0, -30.0));
    mat_Model *= glm::scale(glm::mat4(1.0f), glm::vec3(64.0f,64.0f,0.0f));
    mat_PVM = mat_Proj*mat_View*mat_Model;

   glUniformMatrix4fv(loc_u_pvm_matrix, 1, false, glm::value_ptr(mat_PVM));
   glVertexAttribPointer(loc_a_vertex, 4, GL_FLOAT, false, 0, vertices.data());
   glVertexAttribPointer(loc_a_texcoord, 2, GL_FLOAT, false, 0, texcoords.data());
   glEnableVertexAttribArray(loc_a_vertex);
   glEnableVertexAttribArray(loc_a_texcoord);
   glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
   glDisableVertexAttribArray(loc_a_vertex);
   glDisableVertexAttribArray(loc_a_texcoord);
}
void rightground(){ ... }
void leftground(){ ... }
```

정면 뿐 아니라 좌측, 우측에도 배경을 삽입함

4 마우스(오브젝트 회전)



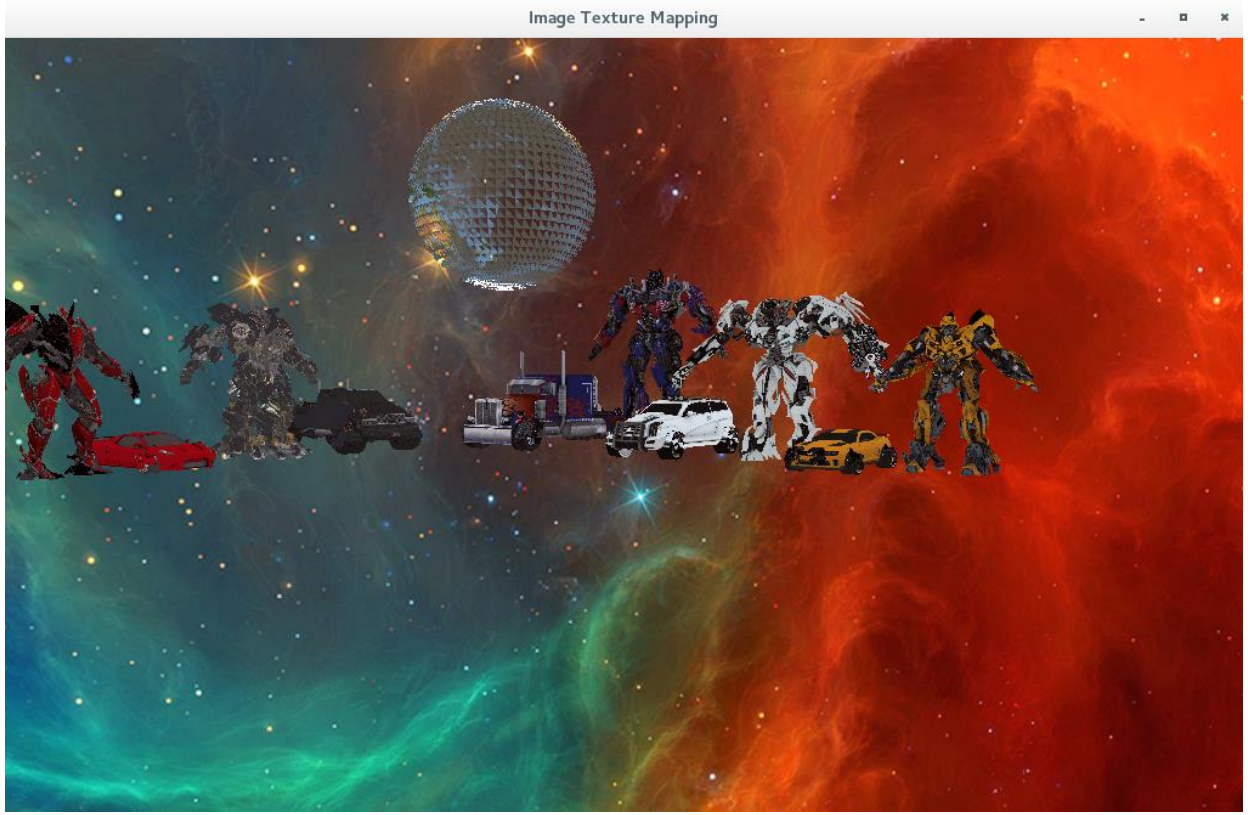



```
void processMouse( int button, int state, int x, int y) {

  glm::vec4 pos;

  specialKey = glutGetModifiers( );
  if ((state == GLUT_DOWN) && (specialKey == GLUT_ACTIVE_ALT)) {

    if (button == GLUT_LEFT_BUTTON)
    {
          obj_rotate_flag = !obj_rotate_flag;
    }
    else if (button == GLUT_RIGHT_BUTTON)
    {
      car_rotate_flag = !car_rotate_flag;
    }
  }
}
```

alt+  => 트랜스포머 오브젝트 5개 회전

alt+  => 자동차 오브젝트 5개 회전

4 마우스(화면 회전)



마우스의 움직임을 감지하여 화면 이동

```
void processMousePassiveMotion(int x, int y)
{

    if(pre_MousePos_x - x < 0)
    {
        g_camera.rotate_left(0.006f);
          pre_MousePos_x = x;

         if(pre_MousePos_y - y <0)
         {
             g_camera.rotate_up(0.003f);
             pre_MousePos_y = y;
         }

         else
         {
              g_camera.rotate_down(0.003f);
              pre_MousePos_y = y;
         }
    }

    else
    {
        g_camera.rotate_right(0.006f);
        pre_MousePos_x = x;

         if(pre_MousePos_y - y <0)
        {
                g_camera.rotate_up(0.003f);
                pre_MousePos_y = y;
         }

         else
        {
                g_camera.rotate_down(0.003f);
                pre_MousePos_y = y;
          }
    }

}
```