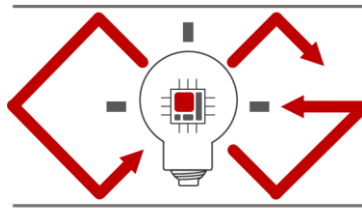


Programming Assignment 3

2023 Computer Graphics



Computer Graphics
Laboratory

Submission

Deadline : 23:59:59, Tuesday, May 23th, 2023 (KST, +0900)

- Github server clock

To submit your assignment, you **must** do two things, **Both of them must be done BEFORE deadline.**

1. You should push your commit to your assignment repo before deadline.
2. You should comment the last commit (before deadline) id (SHA-1 hash) in github issue board. (See next slide)

The last commit **BEFORE** deadline will be considered as submitted assignment.

- Github server will track this for me.
- Timestamp in your commit (local time) will be ignored. (I will use github server timestamp instead)

Policy

In the following cases, your grade for this PA will be 0

- Late submission (Late push before deadline or Late last commit id comment on issue board)
- Build/execution failure
- Making public of your assignment repository

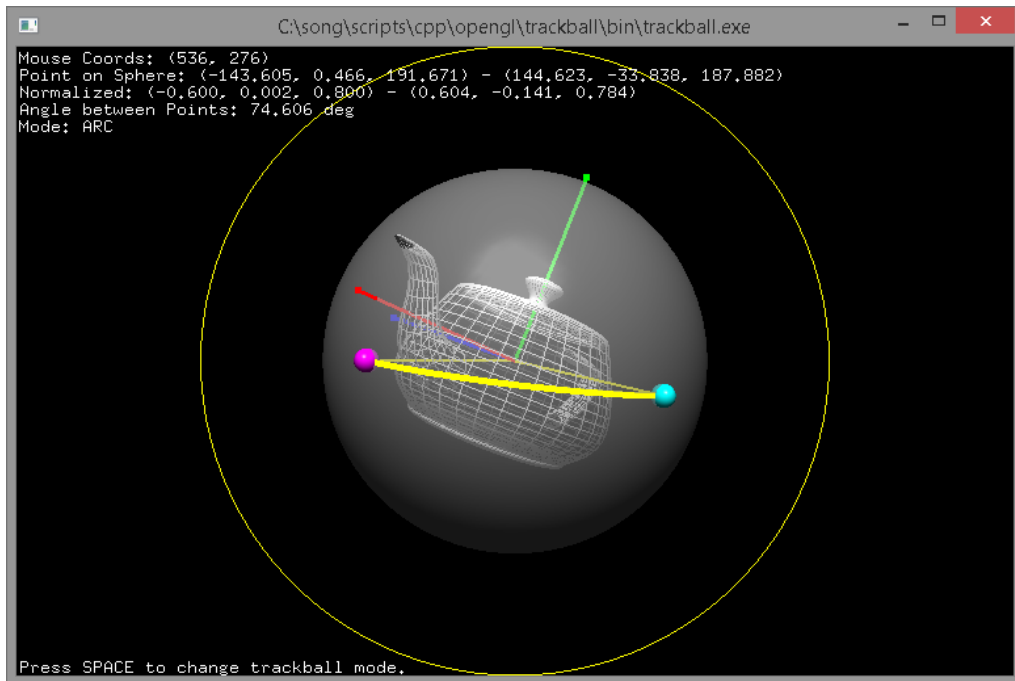
Your final grade will be "F"

- Copy

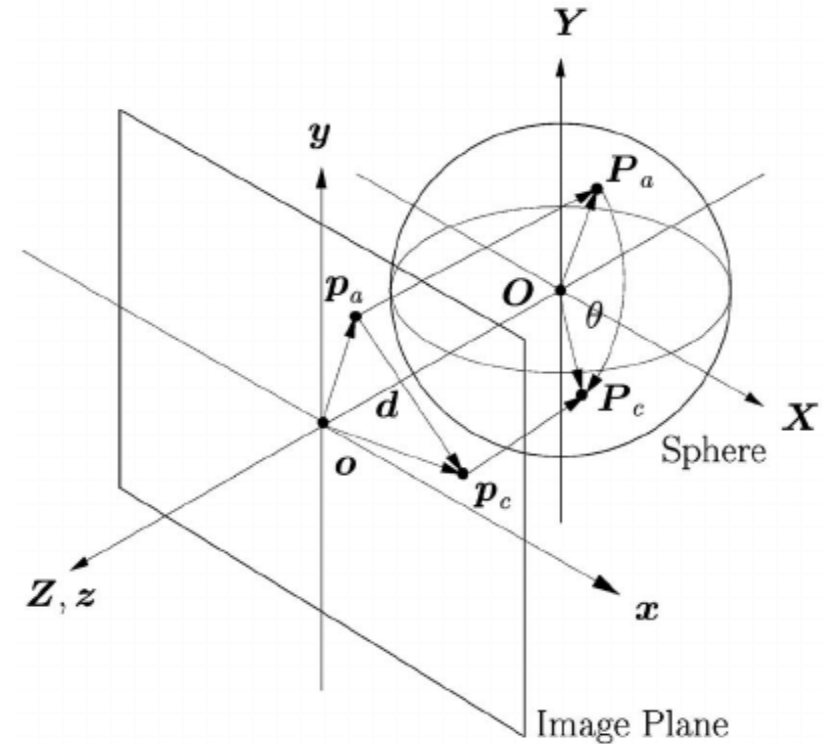
Task Lists

1. Implement trackball camera [18 Points]
2. Lighting [10 Points]
3. Report [2 Points]
 - Write your name, student id, github id in report.md [1 Points]
 - Attach at least two result images in report.md [1 Points]

Trackball Camera



http://www.songho.ca/opengl/gl_camera.html



https://www.researchgate.net/figure/A-virtual-trackball-can-be-thought-of-as-a-3D-sphere-located-behind-the-screen-The_fig2_8329656

The camera which is orbiting around virtual sphere.

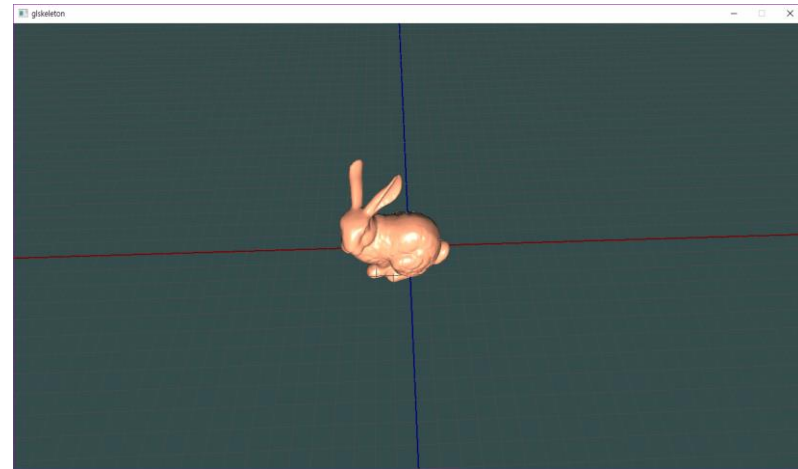
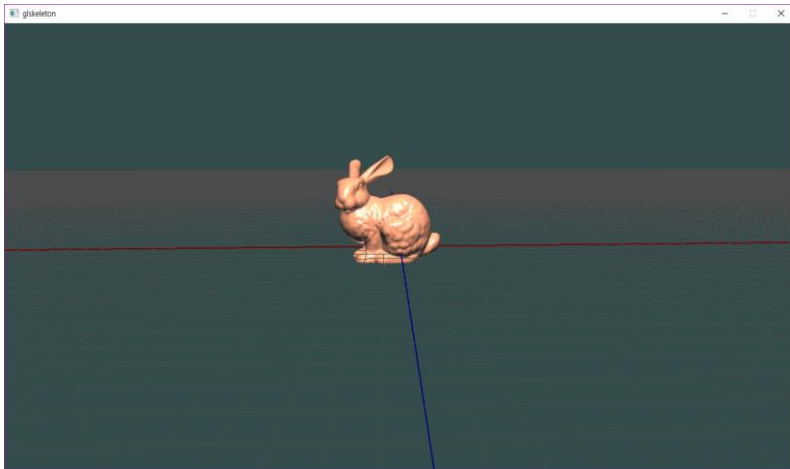
Trackball Camera

How to implement

1. Project your cursors on the sphere
2. Compute rotation matrix with two vectors. (Projected cursor of current and previous frame)
3. Apply the rotation matrix to your camera
 - You need to figure out how to do this.

Task – Trackball Camera

- Fix the lookat position to world **origin**
- Rotate your camera by **dragging** [12 Points]
- Dolly in and dolly out by **scrolling** [3 Points]
 - Move your camera toward/backward to lookat direction.
- Zoom in and zoom out [3 Points]
 - Use key callback to do this. "q" for zoom in, "w" for zoom out



Hint 1. Rotation Between Two Vector

```
#include <glm/gtx/quaternion.hpp>
// reference
// http://www.opengl-tutorial.org/kr/intermediate-tutorials/tutorial-17-quaternions/

glm::quat RotationBetweenVectors(glm::vec3 start, glm::vec3 dest) {
    start = glm::normalize(start);
    dest = glm::normalize(dest);

    float cosTheta = dot(start, dest);
    glm::vec3 rotationAxis;

    if (cosTheta < -1 + 0.001f) {
        // special case when vectors in opposite directions:
        // there is no "ideal" rotation axis
        // So guess one; any will do as long as it's perpendicular to start
        rotationAxis = cross(glm::vec3(0.0f, 0.0f, 1.0f), start);
        if (glm::length2(rotationAxis) <
            0.01) // bad luck, they were parallel, try again!
            rotationAxis = cross(glm::vec3(1.0f, 0.0f, 0.0f), start);

        rotationAxis = normalize(rotationAxis);
        return glm::angleAxis(glm::radians(180.0f), rotationAxis);
    }

    rotationAxis = cross(start, dest);

    float s = sqrt((1 + cosTheta) * 2);
    float invs = 1 / s;

    return glm::quat(s * 0.5f,
                    rotationAxis.x * invs,
                    rotationAxis.y * invs,
                    rotationAxis.z * invs);
}
```

Rotation is usually represented with quaternion.
Understanding quaternion is out of scope.

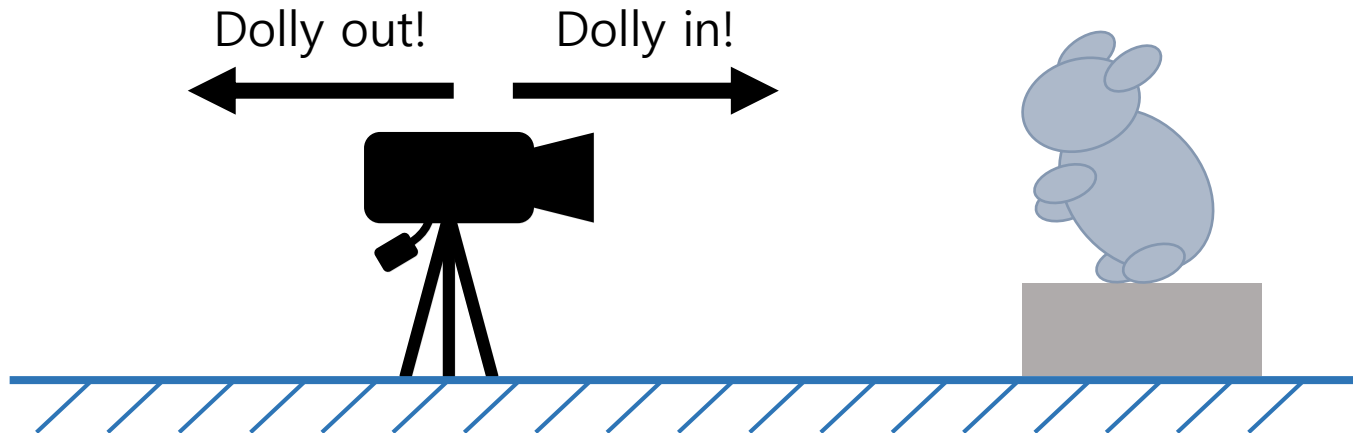
But we can convert them into rotation matrix using
glm.

So use this like this

```
const glm::mat4 R =
    glm::toMat4(RotationBetweenVectors(v1,v2));
```

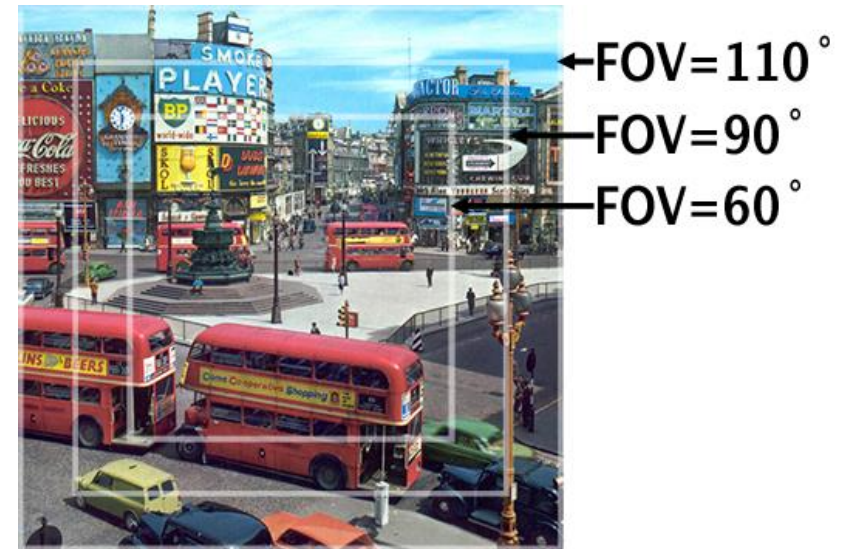
Hint 2. Dolly In/Out

- Translation of camera origin
 - Move your camera toward/backward to look at direction.
- It affects to camera(view) matrix.



Hint 3. Zoom In/Out

- Changing fov of camera
- It affects to projection matrix.

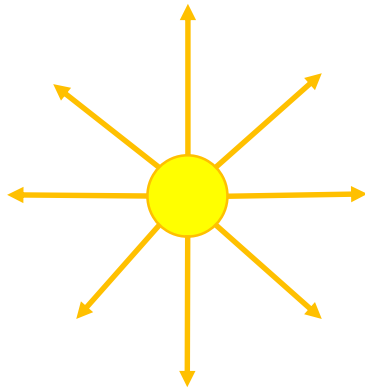


Task - Lighting

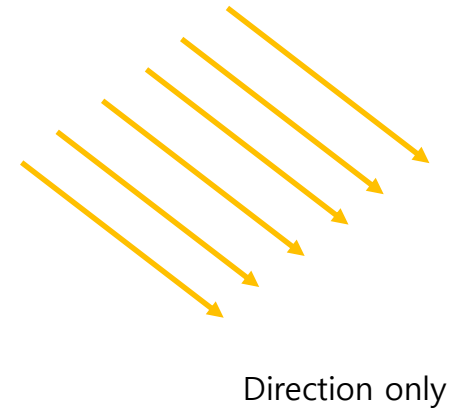
- You should apply phong shading model (ambient/diffuse/specular) or Gouraud shading option.
- Set one point light [5 Point] with on/off functionality.
 - ON, key "1"
 - OFF, key "2"
- Set one directional light [5 Point] with on/off functionality.
 - ON, key "3"
 - OFF, key "4"

Task - Lighting

Point light



Directional light



Commenting Commit ID

master 1 branch 0 tags

Go to file Add file <> Code

| | | |
|----------------|---------------------|-------------|
| ychanu test1 | 444ea6f 3 hours ago | 2 commits |
| doc | Initial commit | 4 hours ago |
| extern | Initial commit | 4 hours ago |
| src | test1 | 3 hours ago |
| .gitignore | Initial commit | 4 hours ago |
| .gitmodules | Initial commit | 4 hours ago |
| CMakeLists.txt | Initial commit | 4 hours ago |
| README.md | Initial commit | 4 hours ago |
| report.md | Initial commit | 4 hours ago |

1. Go to your assignment repository
2. Click commits
3. Click copy button of your last commit

<> Code Issues 1 Pull requests Actions Projects Security Insights

master

Commits on Mar 6, 2023

| | | | |
|----------------|---|------------------|----|
| test1 | ychanu committed 3 hours ago | 444ea6f | <> |
| Initial commit | github-classroom[bot] committed 4 hours ago | Verified 6ba387f | <> |

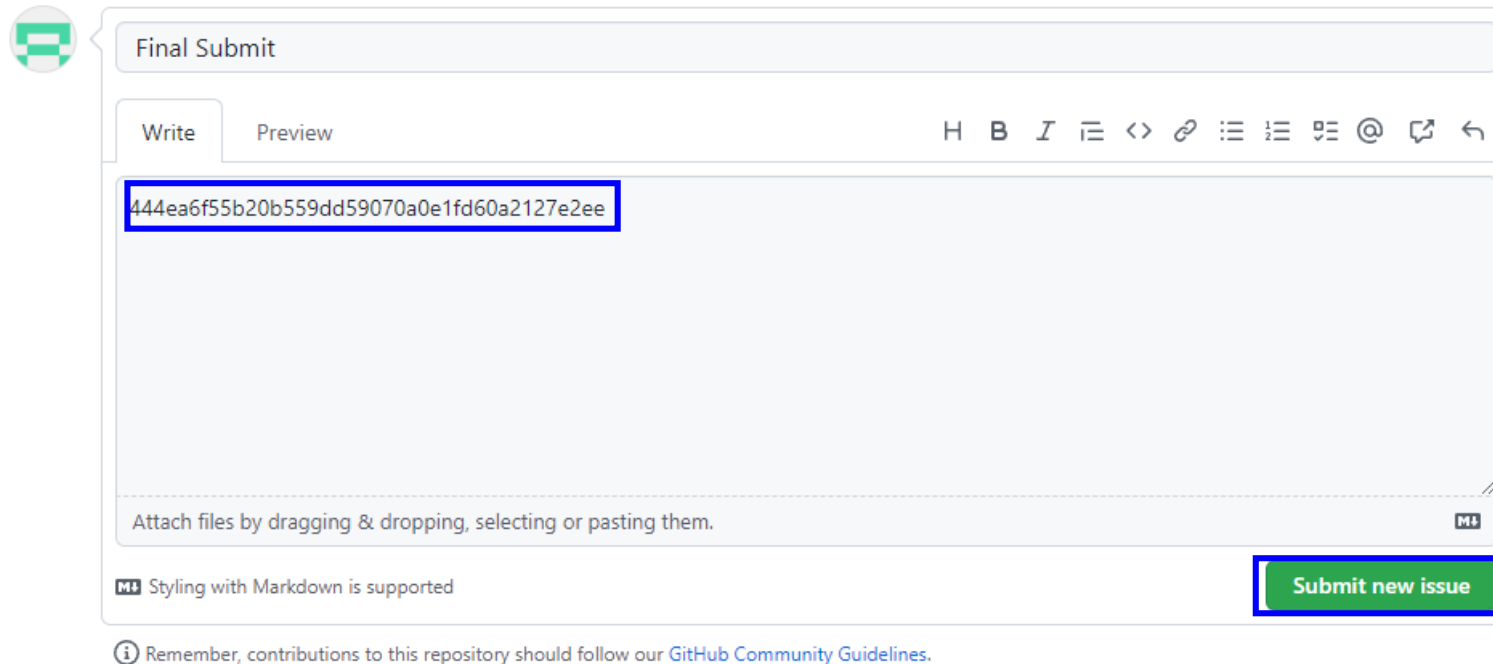
Newer Older

Commenting Commit ID

The screenshot displays the GitHub interface. At the top, the navigation bar includes 'Code', 'Issues 1', 'Pull requests', 'Actions', 'Projects', 'Security', and 'Insights'. The 'Issues 1' tab is highlighted with a blue box. Below the navigation bar, the repository is set to 'master'. The commit history for 'Mar 6, 2023' is shown, with two commits: 'test1' by 'ychanu' (committed 3 hours ago) and 'Initial commit' by 'github-classroom[bot]' (committed 4 hours ago). The commit IDs '444ea6f' and '6ba387f' are visible, along with a 'Verified' badge for the initial commit. Below the commit history, there are 'Newer' and 'Older' buttons. At the bottom, the 'Issues' section is shown with a search filter 'is:issue is:open', 9 labels, and 0 milestones. A 'New issue' button is highlighted with a blue box. The main content area shows '0 Open' and '1 Closed' issues, with a message: 'There aren't any open issues. You could search all of GitHub or try an advanced search.'

1. Go to issue tab
2. Click "New issue"

Commenting Commit ID



The screenshot shows a GitHub issue comment form titled "Final Submit". The form has a "Write" tab selected and a "Preview" tab. The text area contains the commit ID "444ea6f55b20b559dd59070a0e1fd60a2127e2ee", which is highlighted with a blue border. Below the text area, there is a "Submit new issue" button, also highlighted with a blue border. The form includes a rich text editor toolbar with icons for bold, italic, code, link, list, and other formatting options. A footer note reads: "Remember, contributions to this repository should follow our [GitHub Community Guidelines](#)."

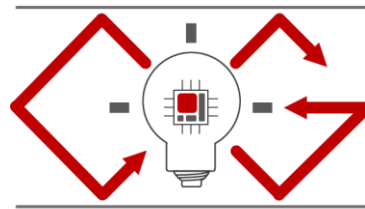
1. Paste your lastest commit id (Ctrl-v)
2. Click "Submit new issue"

PA3 Link

1. Login to github
2. Go to following link <https://classroom.github.com/a/CuwVrVtp>
3. Accept the assignment

Additional Materials for PA

2023 Computer Graphics



Computer Graphics
Laboratory

Additional Materials

If you are having difficulty with the programming assignments, please check the materials as well:

- http://www.songho.ca/opengl/gl_camera.html
- <https://sungcheol-kim.gitbook.io/opengl-tutorial/chapter10>
- <https://cs.lmu.edu/~ray/notes/openglexamples/>

Additional Materials

OpenGL Camera

Under Construction...

Related Topics: [OpenGL Transform](#), [OpenGL Projection Matrix](#), [Quaternion to Rotation Matrix](#)

Download: [OrbitCamera.zip](#), [trackball.zip](#)

- [Overview](#)
- [Camera LookAt](#)
- [Camera Rotation \(Pitch, Yaw, Roll\)](#)
- [Camera Shifting](#)
- [Camera Forwarding](#)
- [Example: Orbit Camera](#)
- [Example: Trackball](#)

Overview

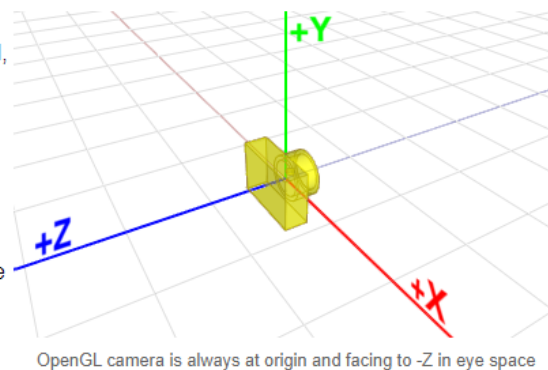
OpenGL doesn't explicitly define neither camera object nor a specific matrix for camera transformation. Instead, OpenGL transforms the entire scene (*including the camera*) inversely to a space, where a fixed camera is at the origin (0,0,0) and always looking along -Z axis. This space is called **eye space**.

Because of this, OpenGL uses a single `GL_MODELVIEW` matrix for both object transformation to world space and camera (view) transformation to eye space.

You may break it down into 2 logical sub matrices;

$$M_{\text{modelView}} = M_{\text{view}} \cdot M_{\text{model}}$$

That is, each object in a scene is transformed with its own M_{model} first, then the entire scene is transformed reversely with M_{view} . In this page, we will discuss only M_{view} for camera transformation in OpenGL.



Screenshot from http://www.songho.ca/opengl/gl_camera.html

LookAt

`gluLookAt()` is used to construct a viewing matrix where a camera is located at the eye position (x_e, y_e, z_e) and looking at (or rotating to) the target point (x_t, y_t, z_t) . The eye position and target are defined in world space. This section describes how to implement the viewing matrix equivalent to `gluLookAt()`.

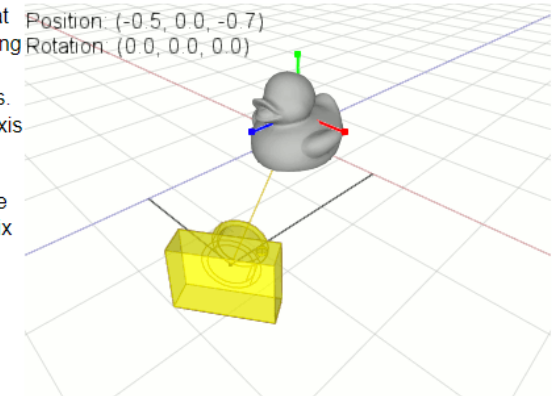
Camera's **lookAt** transformation consists of 2 transformations; translating the whole scene inversely from the eye position to the origin (M_T), and then rotating the scene with reverse orientation (M_R), so the camera is positioned at the origin and facing to the -Z axis.

$$M_{\text{view}} = M_R M_T = \begin{pmatrix} r_0 & r_4 & r_8 & 0 \\ r_1 & r_5 & r_9 & 0 \\ r_2 & r_6 & r_{10} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_0 & r_4 & r_8 & r_0 t_x + r_4 t_y + r_8 t_z \\ r_1 & r_5 & r_9 & r_1 t_x + r_5 t_y + r_9 t_z \\ r_2 & r_6 & r_{10} & r_2 t_x + r_6 t_y + r_{10} t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Suppose a camera is located at (2, 0, 3) and looking at (0, 0, 0) in world space. In order to construct the viewing matrix for this case, we need to translate the world to (-2, 0, -3) and rotate it about -33.7 degree along Y-axis. As a result, the virtual camera becomes facing to -Z axis at the origin.

The translation part of lookAt is easy. You simply move the camera position to the origin. The translation matrix M_T would be (replacing the 4th column with the negated eye position);

$$M_T = \begin{pmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

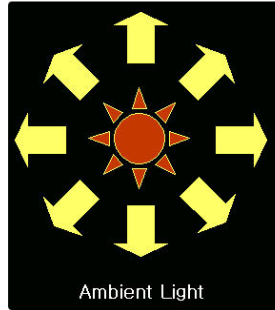


Additional Materials

Screenshot from <https://sungcheol-kim.gitbook.io/opengl-tutorial/chapter10>

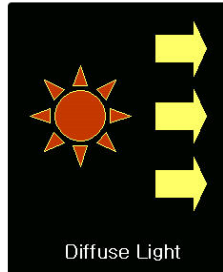
주변광 (Ambient Light)

모든 방향에서 나타나는 조명이다. 즉, 전체적인 분위기를 만들어 내며 단일 지점에서 사방으로 균등하게 광원을 방출하는 조명을 뜻한다. 갓을 씌우지 않은 백열등과 태양이 좋은 예이다.



발산광(Diffuse Light)

이 조명은 한 방향으로부터 나온다. 그리고 특정한 방향과 위치를 가지고 있다. 그러므로 오브젝트의 표면을 비스듬히 비추는 것보다 직각으로 비추는 것이 더욱 강렬한 조명이 된다. 그러나 이 조명의 광선이 오브젝트의 표면에 부딪치면 시야가 어디에 있든지 모든 방향으로 똑같이 반사된다. 테이블을 비추는 램프의 빛을 예로 들 수 있다.



조명 켜기

OpenGL 에서 조명을 사용하려면 다음과 같이 glEnable() 함수를 이용해서 조명에 관련 된 상태 변수를 ON 해줘야한다.

```
glEnable( GL_LIGHTING ); //조명을 사용할 것이다.  
glEnable( GL_LIGHT0 ); //조명 중 0 번 조명을 사용할 것이다.
```

조명을 사용하려면 사용할 조명의 성질을 glLightfv() 함수를 통해서 설정해 주어야 한다.

```
glLightfv( GL_LIGHT0, GL_AMBIENT, AmbientLightValue ); //Ambient 조명의 성질을 설정한다.  
glLightfv( GL_LIGHT0, GL_DIFFUSE, DiffuseLightValue ); //Diffuse 조명의 성질을 설정한다.  
glLightfv( GL_LIGHT0, GL_SPECULAR, SpecularLightValue ); //Specular 조명의 성질을 설정한다.  
glLightfv( GL_LIGHT0, GL_POSITION, PositionLightValue ); //조명의 위치(광원)를 설정한다.
```

위에서 사용하는 각각의 성질값은 아래처럼 정의할 수 있다.

```
GLfloat AmbientLightValue[] = { 0.3f, 0.3f, 0.3f, 1.0f };  
GLfloat DiffuseLightValue[] = { 0.7f, 0.7f, 0.7f, 1.0f };  
GLfloat SpecularLightValue[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
GLfloat PositionLightValue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
```

값을 위처럼 설정할 수도 있지만 조명을 켜기만하면 조명을 사용할 수 있도록 OpenGL 에서는 각각의 성질에 대한 기본 설정값을 가진다. 다른 기본 설정값도 있는데 이 것은 링크 사이트의 OpenGL Programming Guide 사이트를 참고하길 바란다.

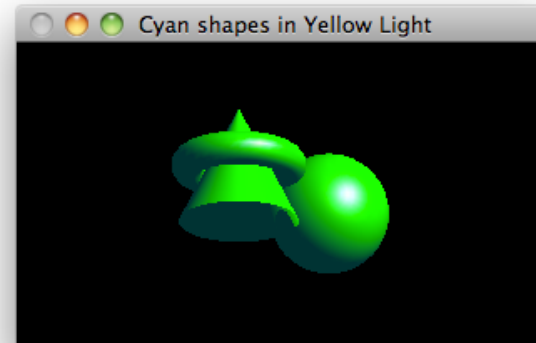
```
GL_AMBIENT (0.0, 0.0, 0.0, 1.0) //Ambient 조명의 기본 설정값  
GL_DIFFUSE (1.0, 1.0, 1.0, 1.0) //Diffuse 조명의 기본 설정값  
GL_SPECULAR (1.0, 1.0, 1.0, 1.0) //Specular 조명의 기본 설정값  
GL_POSITION (0.0, 0.0, 1.0, 0.0) //광원 위치의 기본 설정값  
GL_SPOT_CUTOFF 180.0 //Spot 조명의 Cutoff 기본 설정값
```

Additional Materials

Screenshot from <https://cs.lmu.edu/~ray/notes/openglexamples/>

Lit Solids

- Displays a static picture of three cyan solids lit by a single yellow light source.
- Shows that lighting, like depth buffering, is something that must be enabled.
- Uses ambient, diffuse and specular parameters for lighting.
- Uses a directional light source, as opposed to a point light source.
- Illustrates three of the GLUT easy shape functions.
- Each object is independently moved into the viewing volume, showing the importance of `glPushMatrix` and `glPopMatrix`.



`litsolids.cpp`

```
// This program shows three cyan objects illuminated with a single yellow  
// light source. It illustrates several of the lighting parameters.  
  
#ifdef APPLE_CC
```