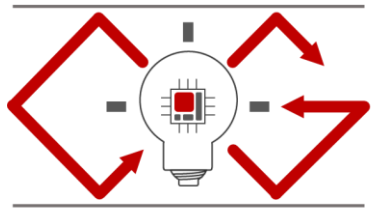
Programming Assignment 3

2022 Computer Graphics



Computer Graphics
Laboratory

Submission

Deadline : 23:59:59, Sunday, May 24th, 2022 (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
- If you tried to push your commit with force option(Tried to change history of remote server)

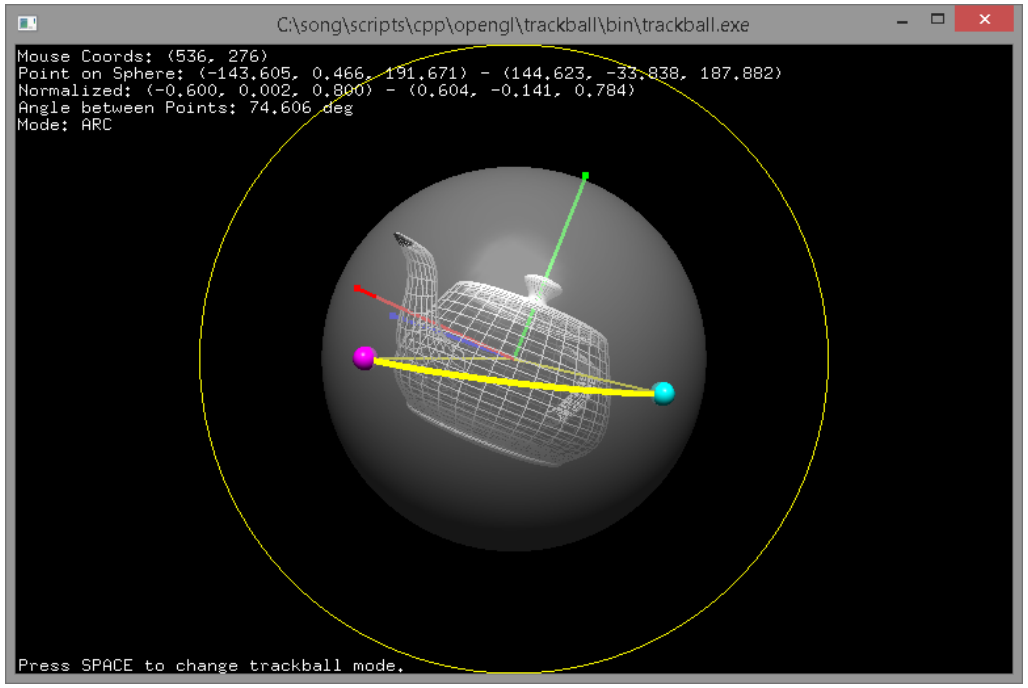
Your final grade will be “F”

- Copy

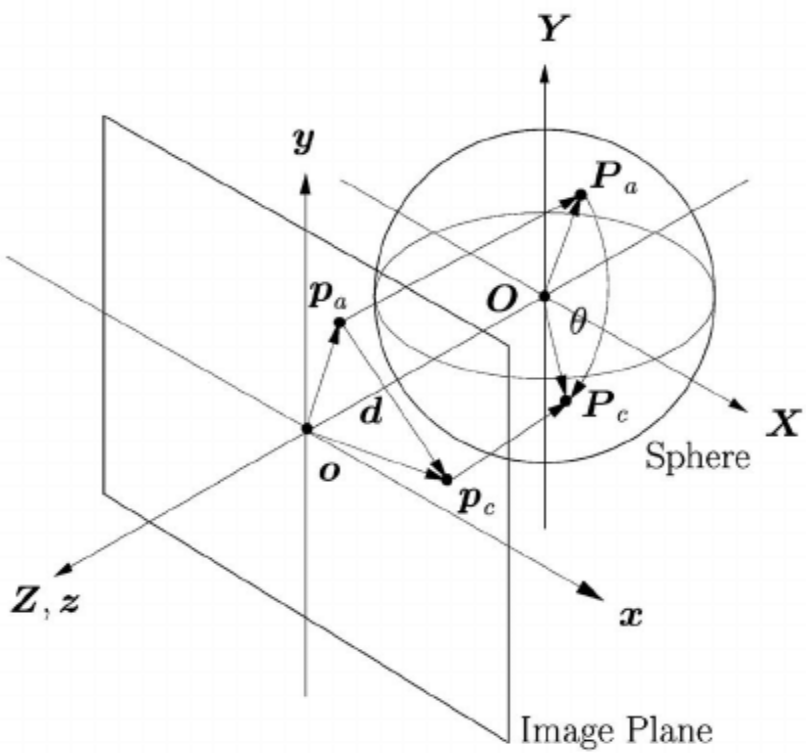
Task Lists

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.

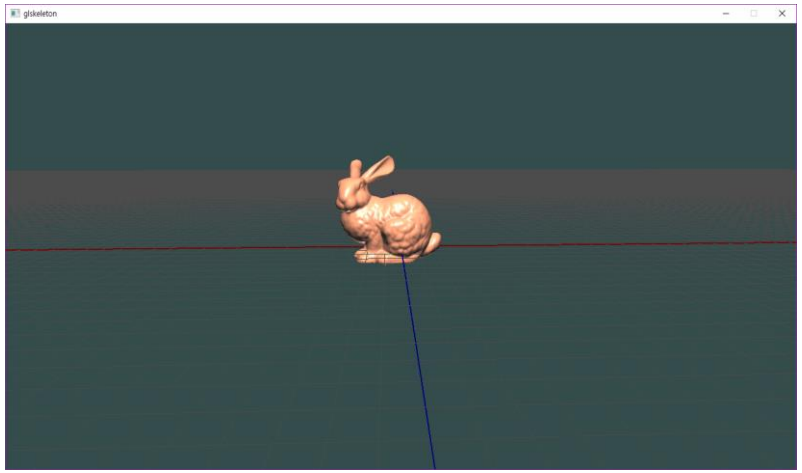
Trackball Camera

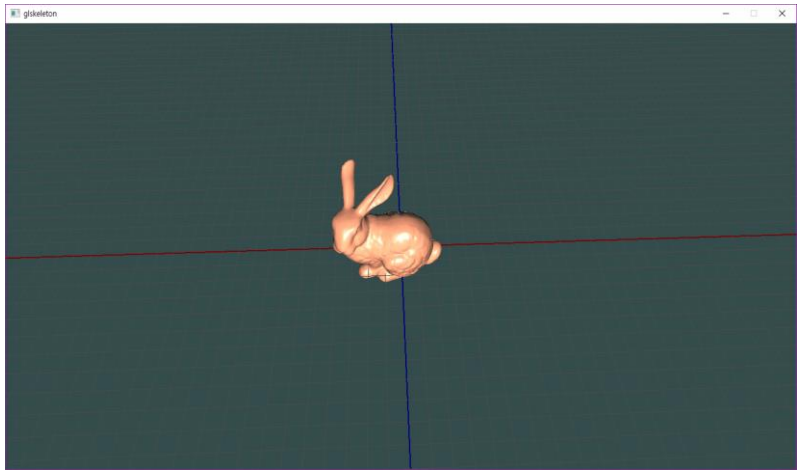
How to implement

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.

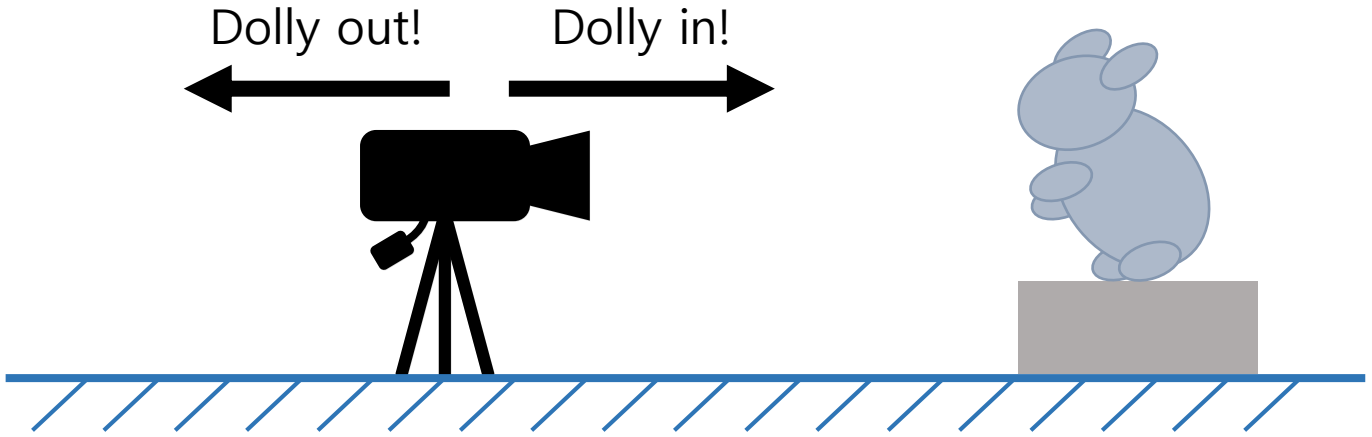
But we can convert them into rotation matrix using glm.

So use this like this

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

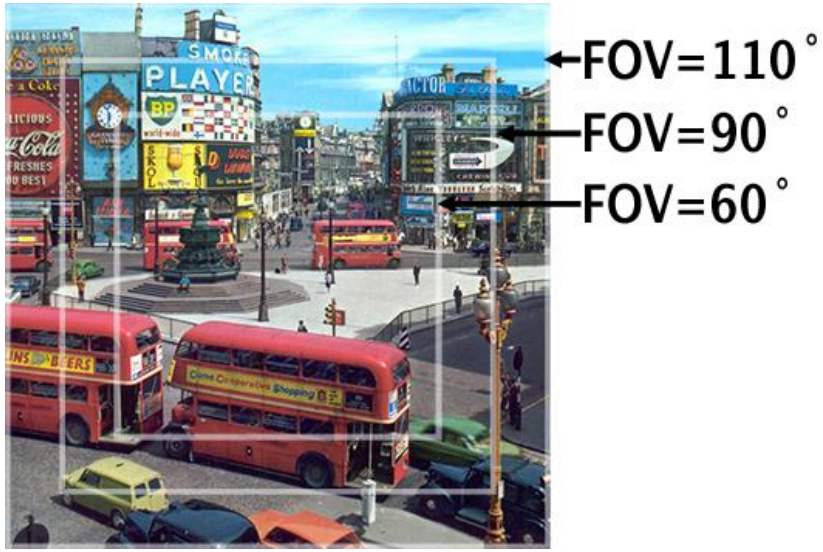
Hint 2. Dolly In/Out

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



Hint 3. Zoom In/Out

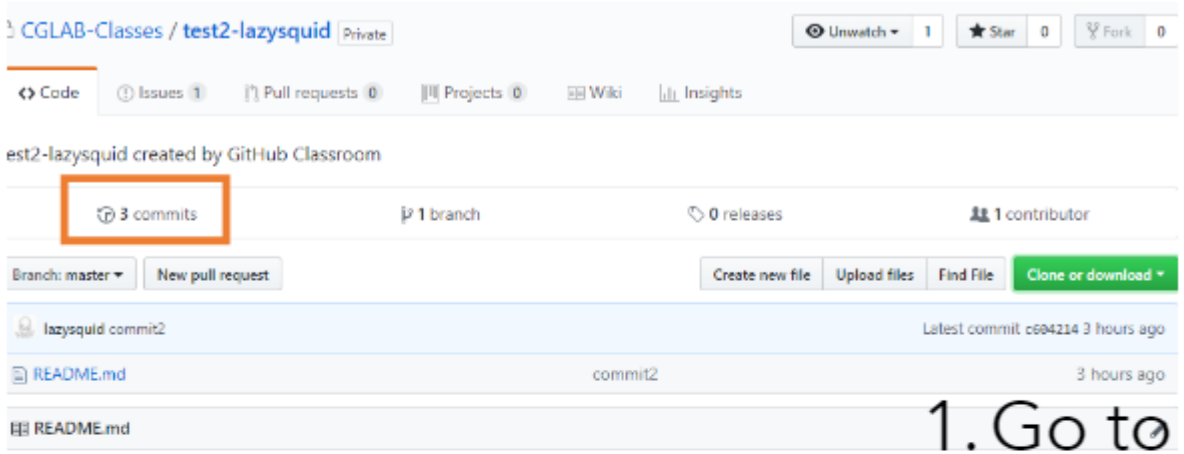
- Changing fov of camera
- It affects to projection matrix.



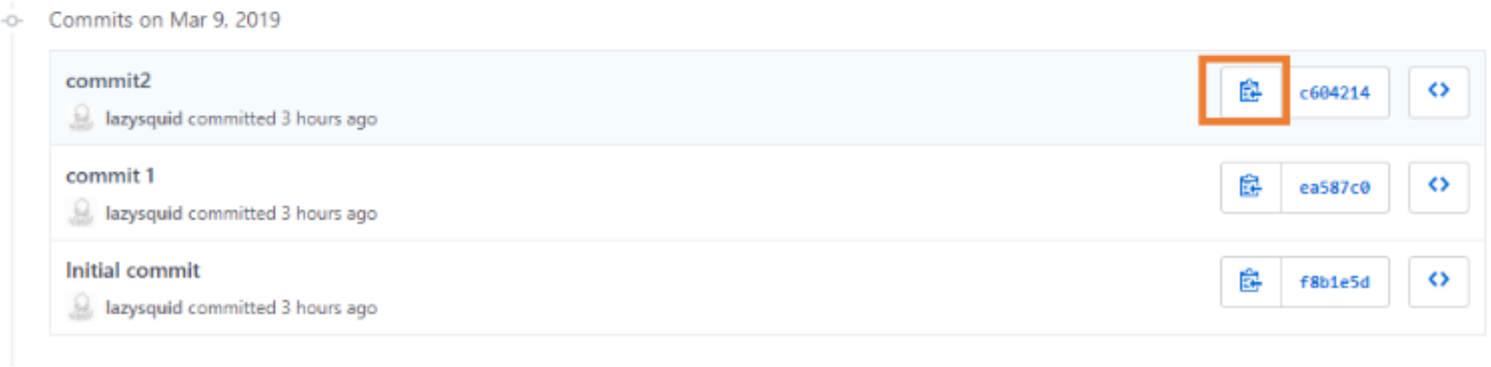
Task - Lighting

- You should apply Phong illumination (ambient/diffuse/specular) with 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”

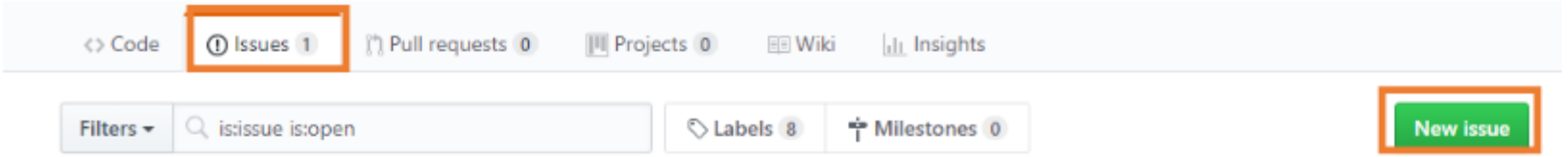
Commenting Commit ID 1/2

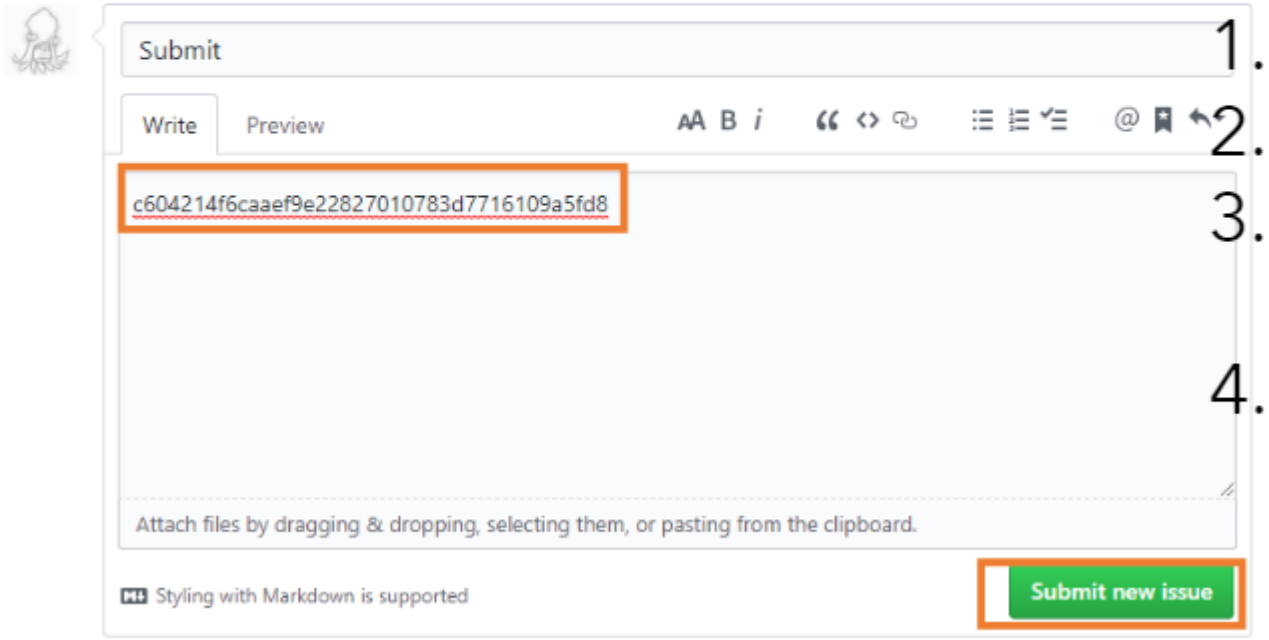


1. Go to your assignment repository
2. Click commits
3. Click copy button of your last commit



Commenting Commit ID 2/2





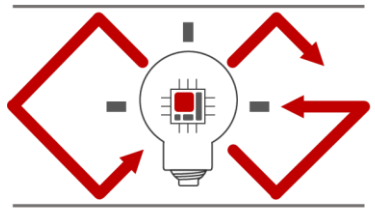
1. Go to issue tab
2. Click “new issue”
3. Paste your lastest commit id (Ctrl-v)
4. Click “submit new isse”

PA3 Link

1. Login to github
2. Go to following link https://classroom.github.com/a/m6ya5Kz5
3. Accept the assignment

Additional Materials for PA

2022 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://tech.burt.pe.kr/opengl/opengl-tutorial
- https://cs.lmu.edu/~ray/notes/openglexamples/

Additional Materials

OpenGL Camera

Under Construction...

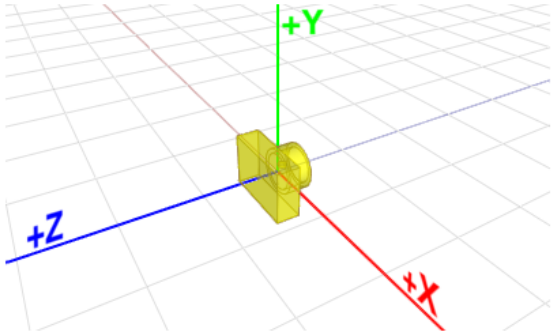
Related Topics: OpenGL Transform, OpenGL Projection Matrix, Quaternion to Rotation Matrix
Download: OrbitCamera.zip, trackball.zip

- 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.



OpenGL camera is always at origin and facing to -Z in 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 $\mathbf{M}_{\text{model}}$ first, then the entire scene is transformed reversely with $\mathbf{M}_{\text{view}}$. In this page, we will discuss only $\mathbf{M}_{\text{view}}$ 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()**.

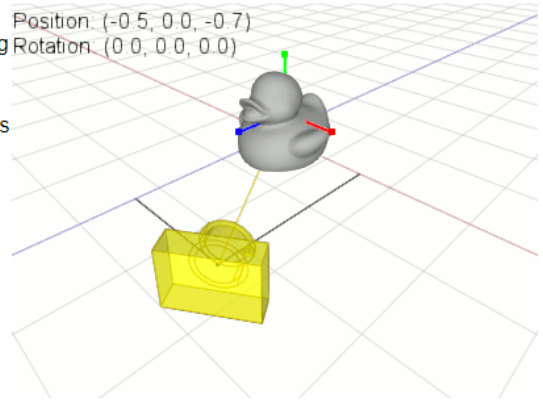
Camera's **lookAt** transformation consists of 2 transformations; translating the whole scene inversely from the eye position to the origin ($\mathbf{M}_T$), and then rotating the scene with reverse orientation ($\mathbf{M}_R$), so the camera is positioned at the origin and facing to the -Z axis.

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

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 $\mathbf{M}_T$ would be (replacing the 4th coloumn with the negated eye position);

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



OpenGL camera's lookAt() transformation

Additional Materials

Screenshot from https://tech.burt.pe.kr/opengl/opengl-tutorial/chapter-11

11장. 빛, 재질 표현하기

https://github.com/skyfe79/OpenGLTutorial

- GL_LIGHTING
- GL_LIGHT
- GL_AMBIENT
- GL_DIFFUSE
- GL_SPECULAR
- GL_POSITION
- GL_SPOT_CUTOFF
- GL_SPOT_DIRECTION
- glEnable()
- glLightfv()
- glNormal
- glMaterialfv()

3D 그래픽스에서 빛이란 없어서는 안 될 존재이다. 3D 그래픽에서 사실감은 무척 중요한 요소인데 이 사실감을 만들어주는 요소 중의 하나가 빛이기 때문이다. OpenGL 은 이렇게 중요한 빛을 몇몇 함수만을 사용하면 표현할 수 있게 아주 쉽게 만들어져 있다. OpenGL 의 빛은 영화 촬영장의 조명을 생각하면 쉽게 이해할 수 있다. 촬영장에서 조명은 배우들만 집중적으로 비추는 것이 있을 것이고, 촬영 배경을 밝게만 해주는 조명 등이 있을 것이다. 이렇게 촬영장에 각각 다른 성질을 부여하여 여러개의 조명을 설치하듯 OpenGL 에서도 여러 성질의 여러개의 조명을 설정할 수 있다. OpenGL 에서는 총 8개의 조명만 설치할 수 있고 각각의 조명이 가질 수 있는 성질에는 주변광, 발산광, 반사광, 스포트라이트가 있다.

조명 켜기

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

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

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

```
1 glLightfv( GL_LIGHT0, GL_AMBIENT, AmbientLightValue ); //Ambient 조명의 성질을 설정
2 glLightfv( GL_LIGHT0, GL_DIFFUSE, DiffuseLightValue ); //Diffuse 조명의 성질을 설정
3 glLightfv( GL_LIGHT0, GL_SPECULAR, SpecularLightValue ); //Specular 조명의 성질을 
4 glLightfv( GL_LIGHT0, GL_POSTION, PotionLightValue ); //조명의 위치(광원)를 설정한다.
```

위에서 사용하는 각각의 성질값은 아래처럼 정의할 수 있다.

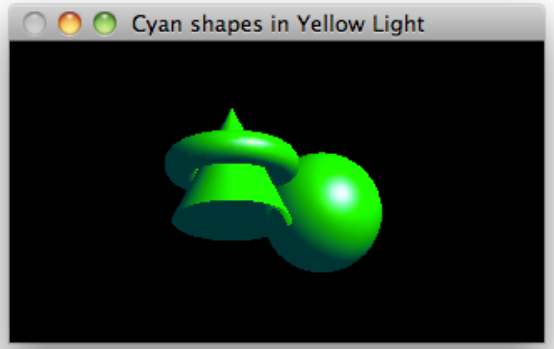
```
1 GLfloat AmbientLightValue[] = { 0.3f, 0.3f, 0.3f, 1.0f };
2 GLfloat DiffuseLightValue[] = { 0.7f, 0.7f, 0.7f, 1.0f };
3 GLfloat SpecularLightValue[] = { 1.0f, 1.0f, 1.0f, 1.0f };
4 GLfloat PositionLightValue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
```

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__
```