CENG 477
Introduction to Computer Graphics

OpenGL Introduction

Yusuf Mücahit Çetinkaya

What is OpenGL?

- Considered as API
- Merely a specification
  - Developed by Khronos Group

- Most implementations are done by graphics card manufacturers.

- Varying libraries with different levels of abstraction: GL, GLU, GLAD, GLFW, GLUT etc.

OpenGL libraries

- **GL & GLEW**
  - Lowest level definitions: vertex, matrix multiplication etc.
  - glVertex3f(point.x, point.y, point.z)
- **GLU**
  - Helper functions for shapes, transformations
  - gluPerspective(fovy, aspect, near, far)
- **GLUT (old)**
  - Highest level: Window and interface management
  - GlutSwapBuffers() // Remember double buffering
- **GLFW (new)**
  - Window and interface management
  - More flexible

OpenGL Implementations

- Simply import libraries
  #include <GL/glew.h>
  #include <GLFW/glfw3.h>

- OSs all provide platform specific implementation.
  Windows: opengl32.lib glu32.lib glfw32.lib
  Linux: -lGL, -lGLU, -lGLEW, -lglfw3

OpenGL Conventions

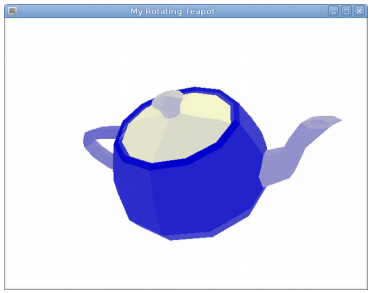
- Many functions have multiple forms:
  - glVertex2f, glVertex3i, glVertex4dv etc.
- Number indicates dimension.
- Letter indicates data type.
  f: float, d: double, i: integer etc.
- ‘v’ indicates a single pointer argument
  glVertex3f(point.x, point.y, point.z)
  glVertex3fv(point)

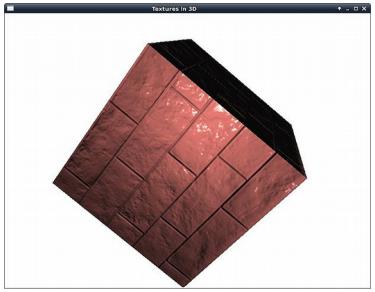
OpenGL API

As a developer, simply do the followings to get image:

1) Specify the location/parameters of camera.
2) Specify the geometry.
3) Specify the lights.

OpenGL will compute the resulting 3D image!



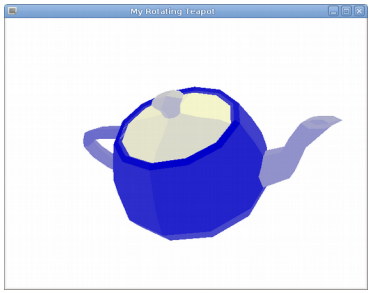


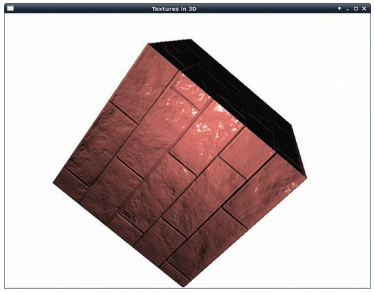
OpenGL API

As a developer, simply do the followings to get image:

1) Specify the location/parameters of camera.
2) Specify the geometry.
3) Specify the lights.

OpenGL will compute the resulting 3D image!





OpenGL Camera

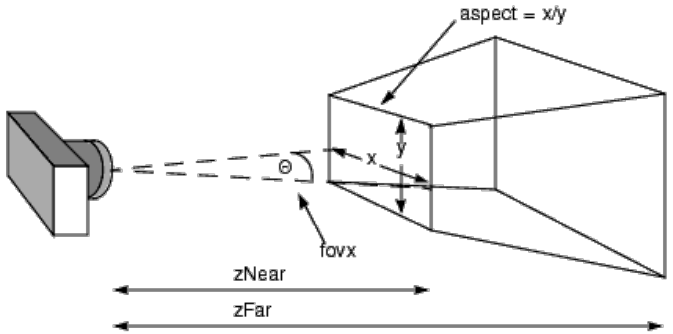
1) Specify the location/parameters of camera.
   a) Physical location of camera
      Where is camera? (eyex, eyey,eyez)
      Which direction is it pointing? (centerx, centery, centerz)
      What is the orientation?(upx, upy, upz)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyex,eyey,eyez,centerx,centery,centerz,upx,upy,upz);
```

OpenGL Camera

1) Specify the location/parameters of camera.
    b) Projection properties of the camera
        Depth of field?
        Field of view in x and y directions?

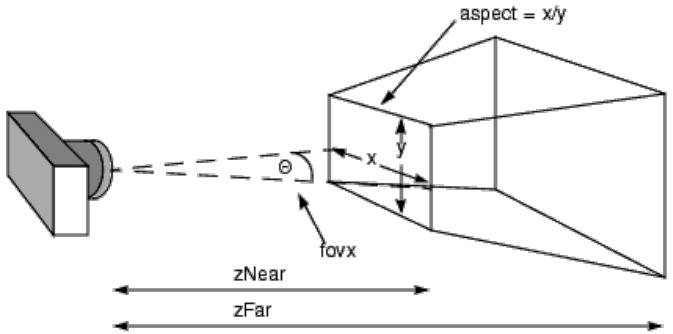
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovy, aspect, near, far);
```



OpenGL Camera

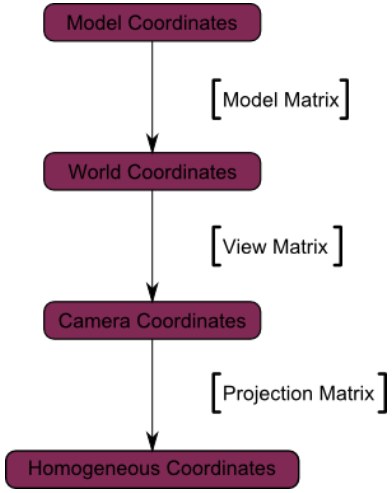
1) Specify the location/parameters of camera.
    b) Projection properties of the camera
        Depth of field?
        Field of view in x and y directions?

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(near_plane.x1, near_plane.x2, near_plane.y1,
near_plane.y2, near_distance, far_distance);
```

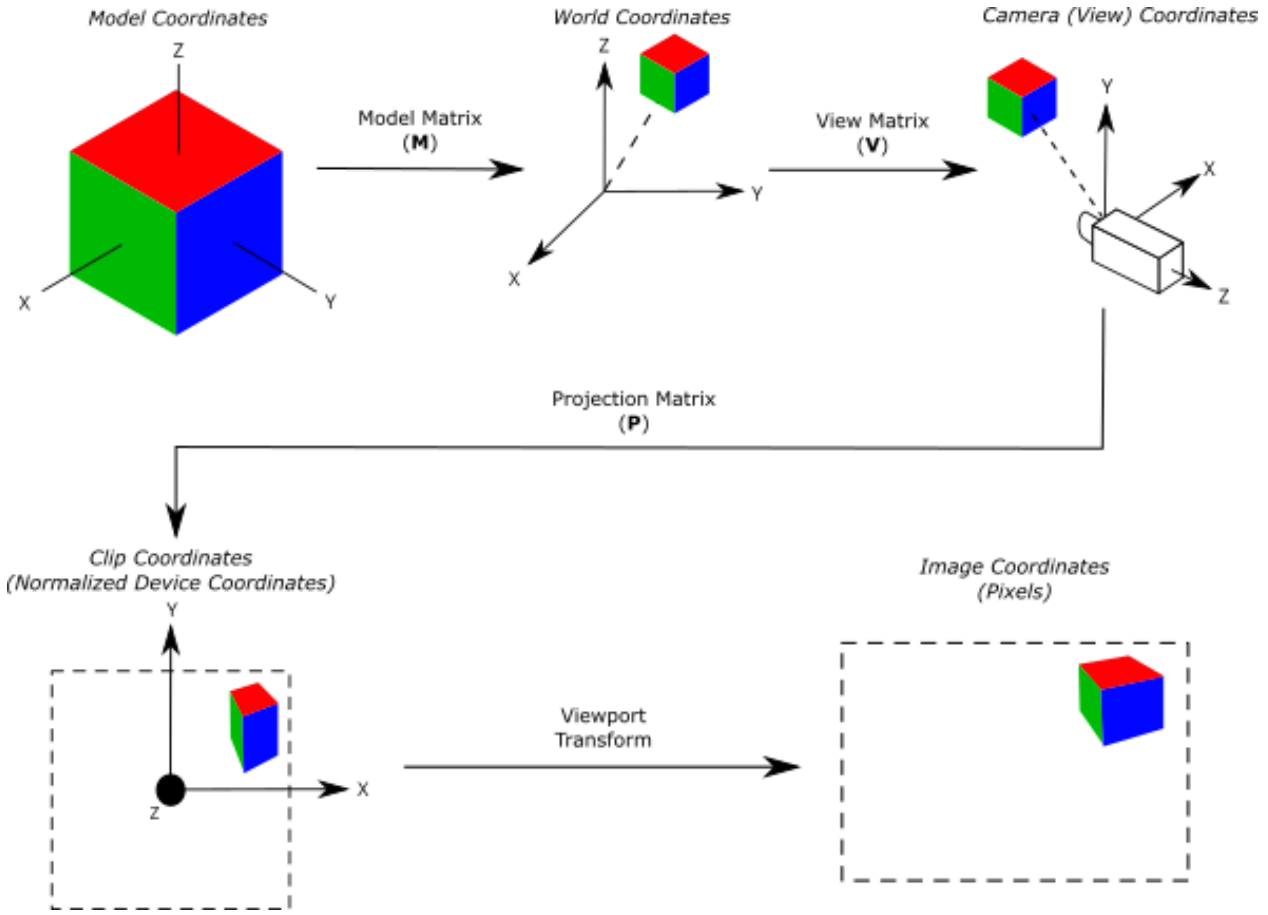


OpenGL Camera

- Coordinate-systems are represented as matrices in OpenGL.
- Think of camera as implying a coordinate-system centered at its image plane.
- Therefore, specifying this matrix implies specifying the physical properties of the camera.

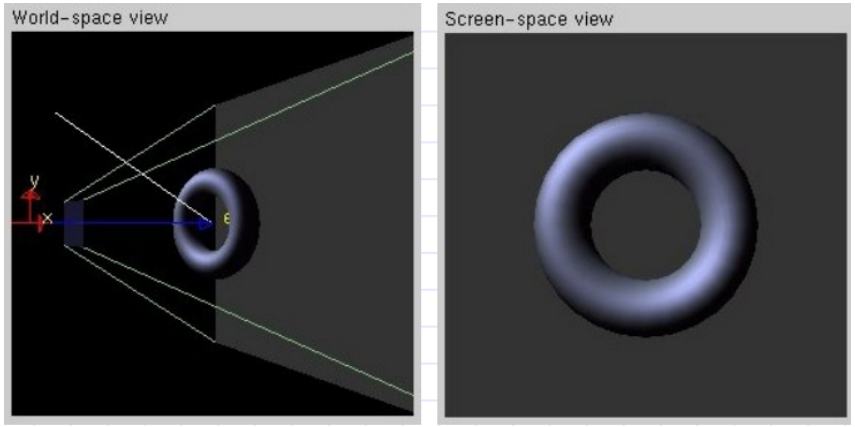


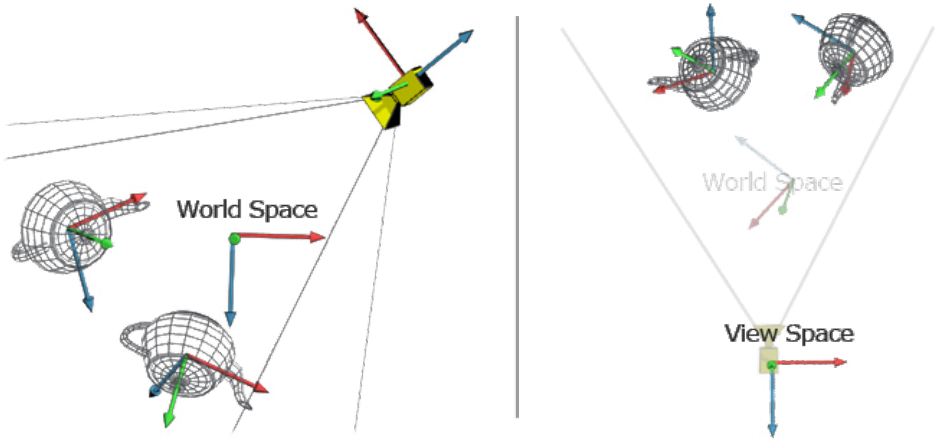
OpenGL Camera



OpenGL: MODELVIEW

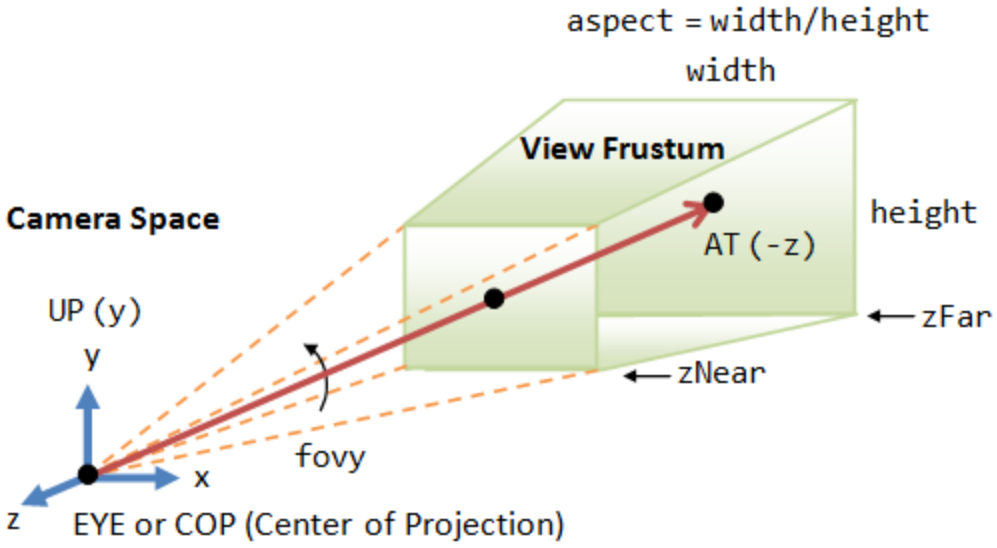
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyex,eyey,eyez,centerx,centery,centerz,upx,upy,upz);
```





OpenGL: PROJECTION

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovy, aspect, near, far);
```



Perspective Projection: The camera's view frustum is specified via 4 view parameters: fovy, aspect, zNear and zFar.

OpenGL Camera

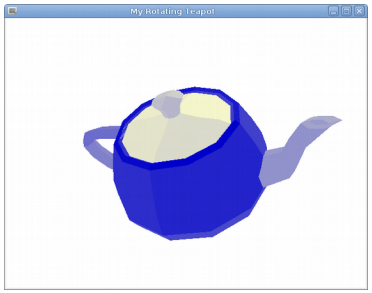
```
void setCamera() {
	glViewport(0, 0, width, height);
	/* Set camera position */
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(m_vEye[0], m_vEye[1], m_vEye[2],
		m_vRef[0], m_vRef[1], m_vRef[2],
		m_vUp[0], m_vUp[1], m_vUp[2]);
	/* Set projection frustrum */
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glFrustum(near_plane.x1, near_plane.x2, near_plane.y1,
		near_plane.y2, near_distance, far_distance);
}
```

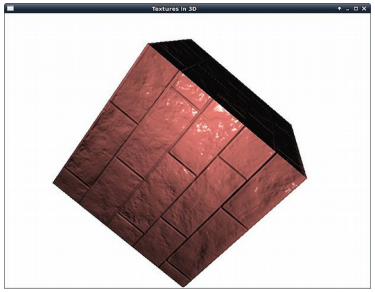
OpenGL API

As a developer, simply do the followings to get image:

1) Specify the location/parameters of camera.
2) Specify the geometry.
3) Specify the lights.

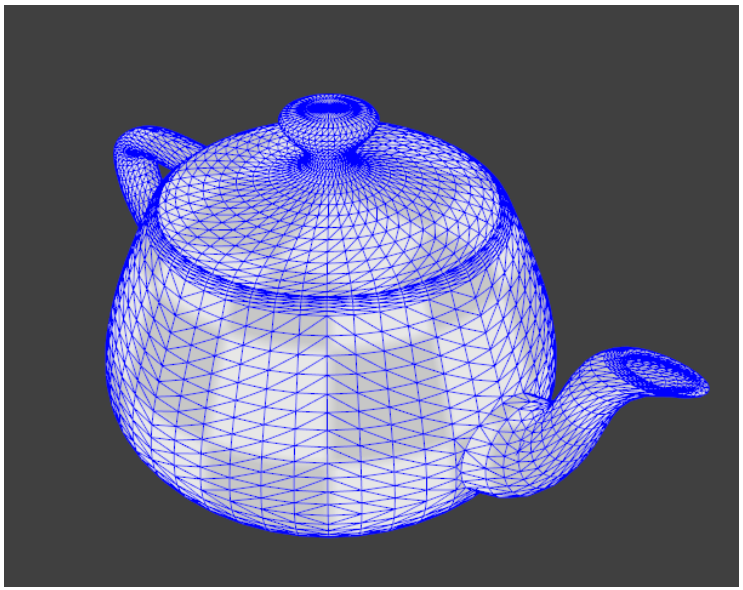
OpenGL will compute the resulting 3D image!





OpenGL Draw

2) Specify the geometry.
   Using primitives: triangles, quadrilaterals, lines etc.



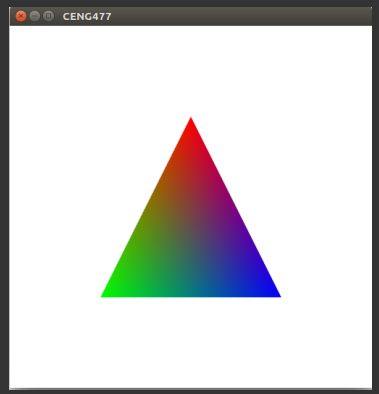
OpenGL Draw

```
void drawObject() {
	glBegin(GL_TRIANGLES);
	for (int i = 0; i < ntris; i++) {
		glColor3f(tri[i].r0, tri[i].g0, tri[i].b0); // Color of vertex
		glNormal3f(tri[i].nx0, tri[i].ny0, tri[i].nz0); // Normal of
vertex
		glVertex3f(tri[i].x0, tri[i].y0, tri[i].z0); // Position of vertex
		…
		glColor3f(tri[i].r2, tri[i].g2, tri[i].b2);
		glNormal3f(tri[i].nx2, tri[i].ny2, tri[i].nz2);
		glVertex3f(tri[i].x2, tri[i].y2, tri[i].z2);
	}
	glEnd(); // Sends all the vertices/normals to the OpenGL
library
}
```

OpenGL Draw

- OpenGL is a state machine!

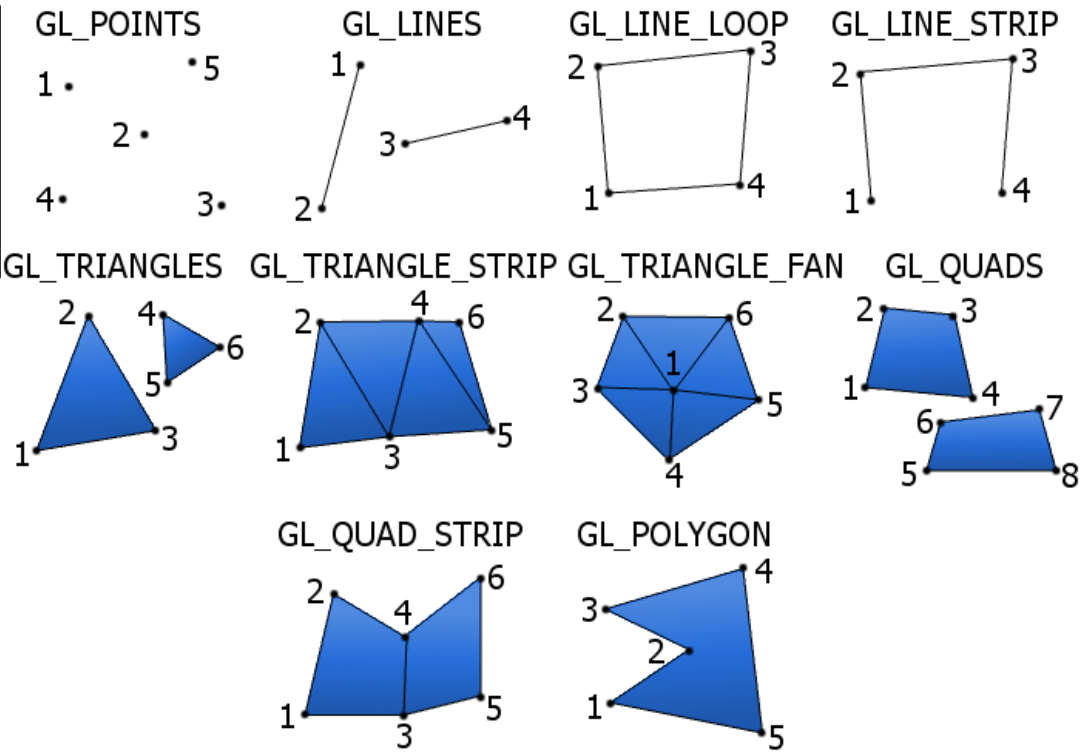
```
glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0); // color state is red
glVertex3f(0, 0.5, 0); // first vertex
glColor3f(0.0, 1.0, 0.0); // color state is green
glVertex3f(-0.5, -0.5, 0); // second vertex
glColor3f(0.0, 0.0, 1.0); //color state is blue
glVertex3f(0.5, -0.5, 0); // third vertex
glEnd();
```



OpenGL Draw

- OpenGL supports many primitives.

```
glBegin(GL_TRIANGLES);
glBegin(GL_QUADS);
glBegin(GL_POLYGON);
...
```



OpenGL Transformations

- OpenGL matrix transformations:

```
void glTranslatef(GLfloat x, GLfloat y, Glfloat z);
void glRotatef(GLdouble angle, GLfloat x, GLfloat y, GLfloat z);
void glScalef(GLfloat x, GLfloat y, Glfloat z);
```

OpenGL Transformations

- glPushMatrix() and glPopMatrix()
  - Push and pop into current matrix stack.
- They are used to transform a specific object in an OpenGL scene:

```
glPushMatrix();
glTranslatef(GLfloat x, GLfloat y, Glfloat z);
glRotatef(GLdouble angle, GLfloat x, GLfloat y, GLfloat z);
glScalef(GLfloat x, GLfloat y, Glfloat z);
...
drawObjects();
glPopMatrix();
```

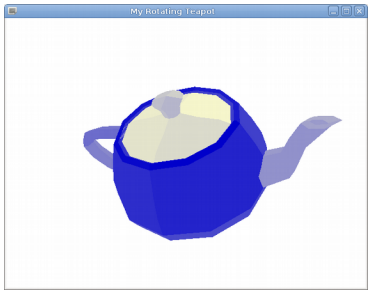
Note: The order is important! It is reversed like transformation matrix multiplication.

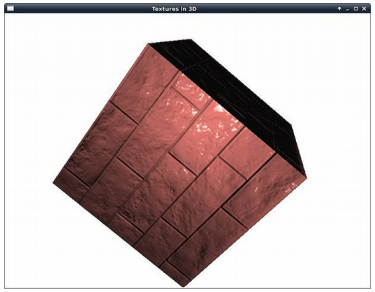
OpenGL API

As a developer, simply do the followings to get image:

1) Specify the location/parameters of camera.
2) Specify the geometry.
3) Specify the lights.

OpenGL will compute the resulting 3D image!

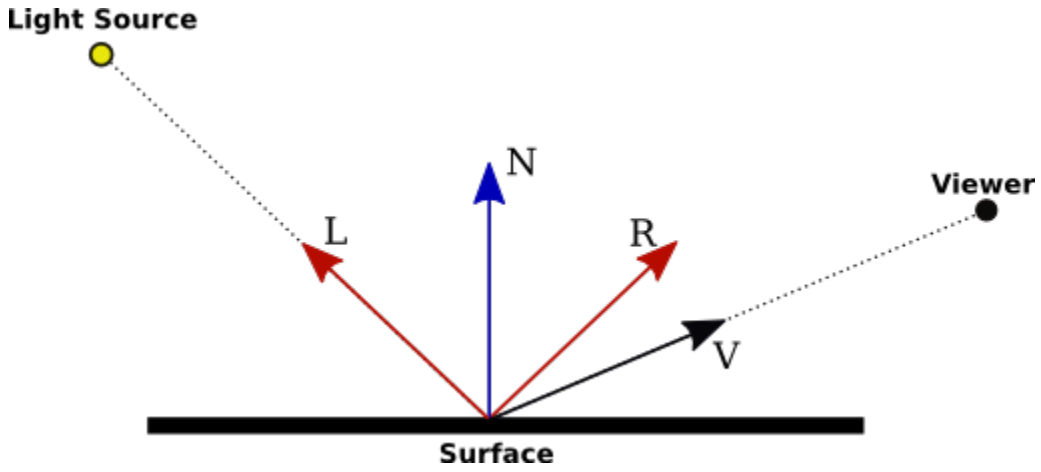




OpenGL Lighting

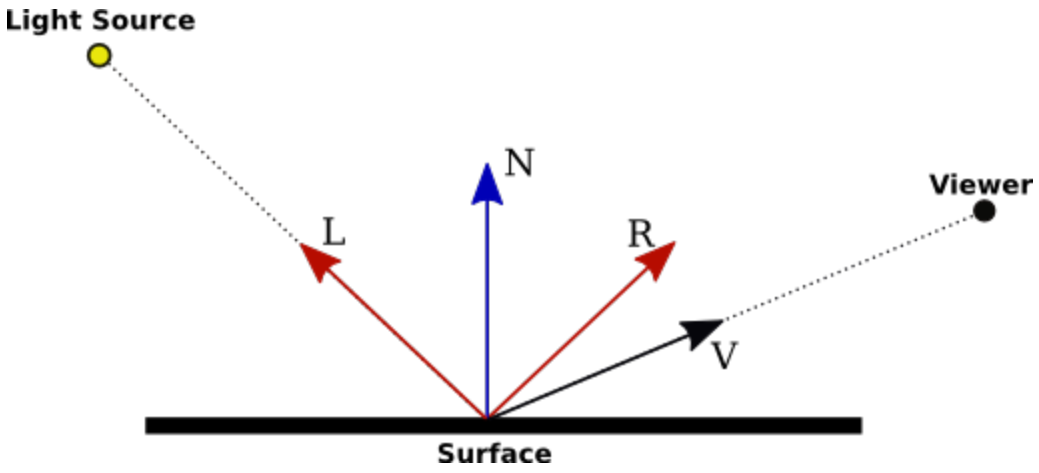
3) Specify the lights.

- OpenGL computes the color at each vertex with a shading calculation:



OpenGL Lighting

```
color = ambient;
for (int i = 0; i < nlights; i++) {
      color += (diffuse + specular * dot(n, h) shine) * cos(θin) * light_color[i];
```



OpenGL Lighting

- Lighting is optional.
- When disabled, the color of each vertex is directly glColor3f(r,g,b)

```
glEnable(GL_LIGHTING);
glDisable(GL_LIGHTING);
```

OpenGL Lighting

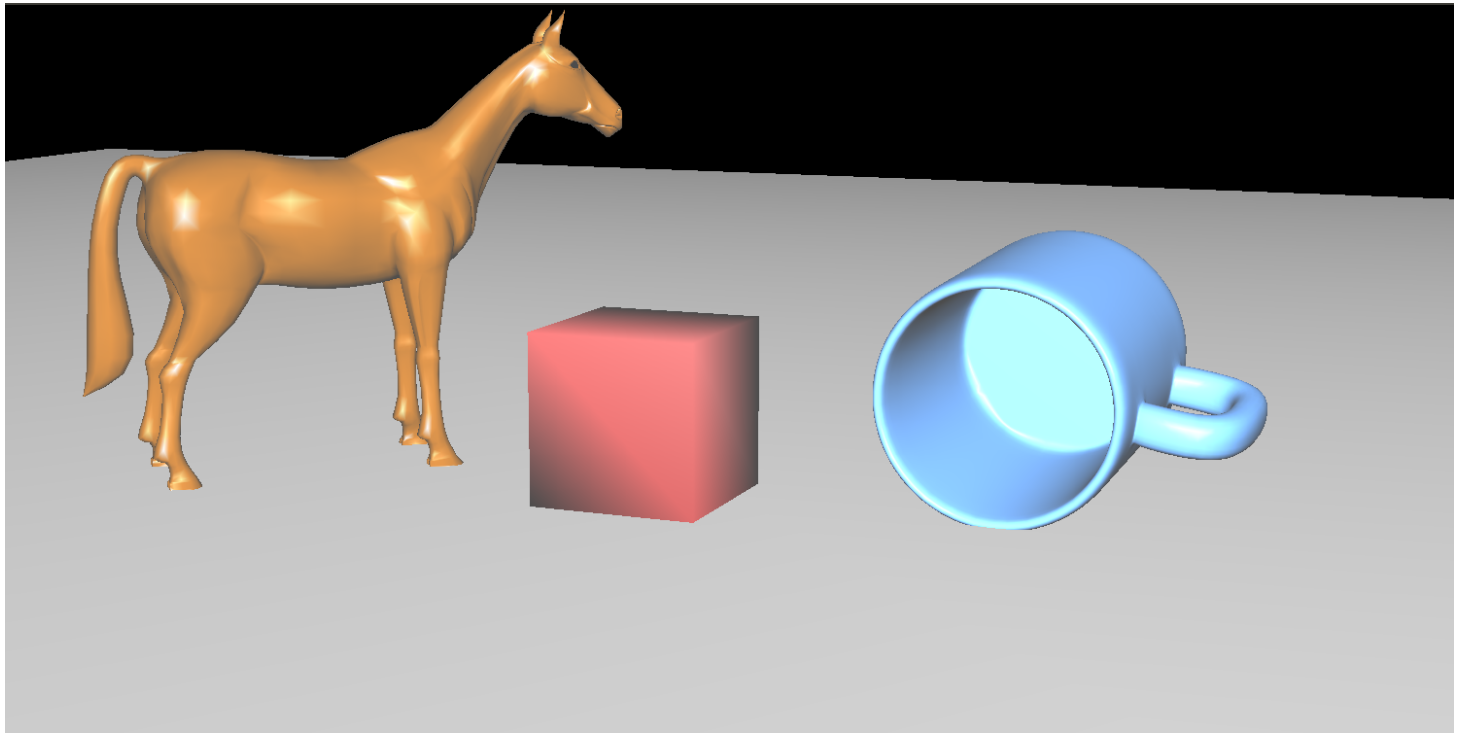
```
void turnOnLights() {
	glEnable(GL_LIGHTING);
	for (int i = 0; i < nlights; i++) {
		glEnable(GL_LIGHT0 + I);
		GLfloat col[] = {lights[i].intensity.R, lights[i].intensity.G, lights[i].intensity.B, 1.0f};
		GLfloat pos[] = {lights[i].position.X, lights[i].position.Y, lights[i].position.Z, 1.0f};

		glLightfv(GL_LIGHT0 + i, GL_POSITION, pos);
		glLightfv(GL_LIGHT0 + i, GL_AMBIENT, ambient);
		glLightfv(GL_LIGHT0 + i, GL_DIFFUSE, col);
		glLightfv(GL_LIGHT0 + i, GL_SPECULAR, col);
	}
}

void turnOffLights() {
	glDisable(GL_LIGHTING);
	for (int i = 0; i < nlights; i++)
		glDisable(GL_LIGHT0 + I);
}
```

OpenGL API

- OpenGL will compute the resulting 3D image!



GLFW Library

- GLFW is a free, Open Source, multi-platform library for **OpenGL**, **OpenGL ES** and **Vulkan** application development.
- It provides a simple, platform-independent **API** for **creating windows, contexts and surfaces, reading input, handling events,** etc.

GLFW Library

- Include only GLFW! Do not include OpenGL header!
  GLFW does it for you.
- Customized headers should be included before GLFW.

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
```

GLFW Library

- Define error callback function for debugging.

```
static void errorCallback(int error, const char* description)
{
    fprintf(stderr, "Error: %s\n", description);
}
...
glfwSetErrorCallback(errorCallback);
```

GLFW Library

- Initialize GLFW.
  - If fails return!

```
if (!glfwInit()) {
        exit(EXIT_FAILURE);
}
```

- Create window

```
static GLFWwindow* win = NULL;
win = glfwCreateWindow(width, height, "CENG477", NULL, NULL);
if (!win) {
        glfwTerminate();
        exit(EXIT_FAILURE);
}
```

GLFW Library

- Define key callback function for user interaction.

```
static void keyCallback(GLFWwindow* window, int key, int scancode, int
action, int mods) {
	if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
		glfwSetWindowShouldClose(window, GLFW_TRUE);
}
...
glfwSetKeyCallback(win, keyCallback);
...
glfwMakeContextCurrent(win); // Register opengl context to windows
```

GLFW Library

- Do rendering while window is not closed.
  - GLFW checks it automatically.

```
...
while (!glfwWindowShouldClose(win)) {
        glfwWaitEvents();
        customizedRenderFunction();
        glfwSwapBuffers(win);
}

glfwDestroyWindow(win);
glfwTerminate();
...
```

GLFW Library

- Sample customized render function:

```
void customizedRenderFunction() {
	glBegin(GL_TRIANGLES);
	glColor3f(1.0, 0.0, 0.0); // color state is red
	glVertex3f(0, 0.5, 0); // first vertex
	glColor3f(0.0, 1.0, 0.0); // color state is green
	glVertex3f(-0.5, -0.5, 0); // second vertex
	glColor3f(0.0, 0.0, 1.0); //color state is blue
	glVertex3f(0.5, -0.5, 0); // third vertex
	glEnd();
}
```

- Some useful resources:
- https://learnopengl.com/
- http://www.glfw.org/docs/latest/quick.html
- http://www.opengl-tutorial.org/beginners-tutorials/