Grafika Komputer

OpenGL 4
Viewing

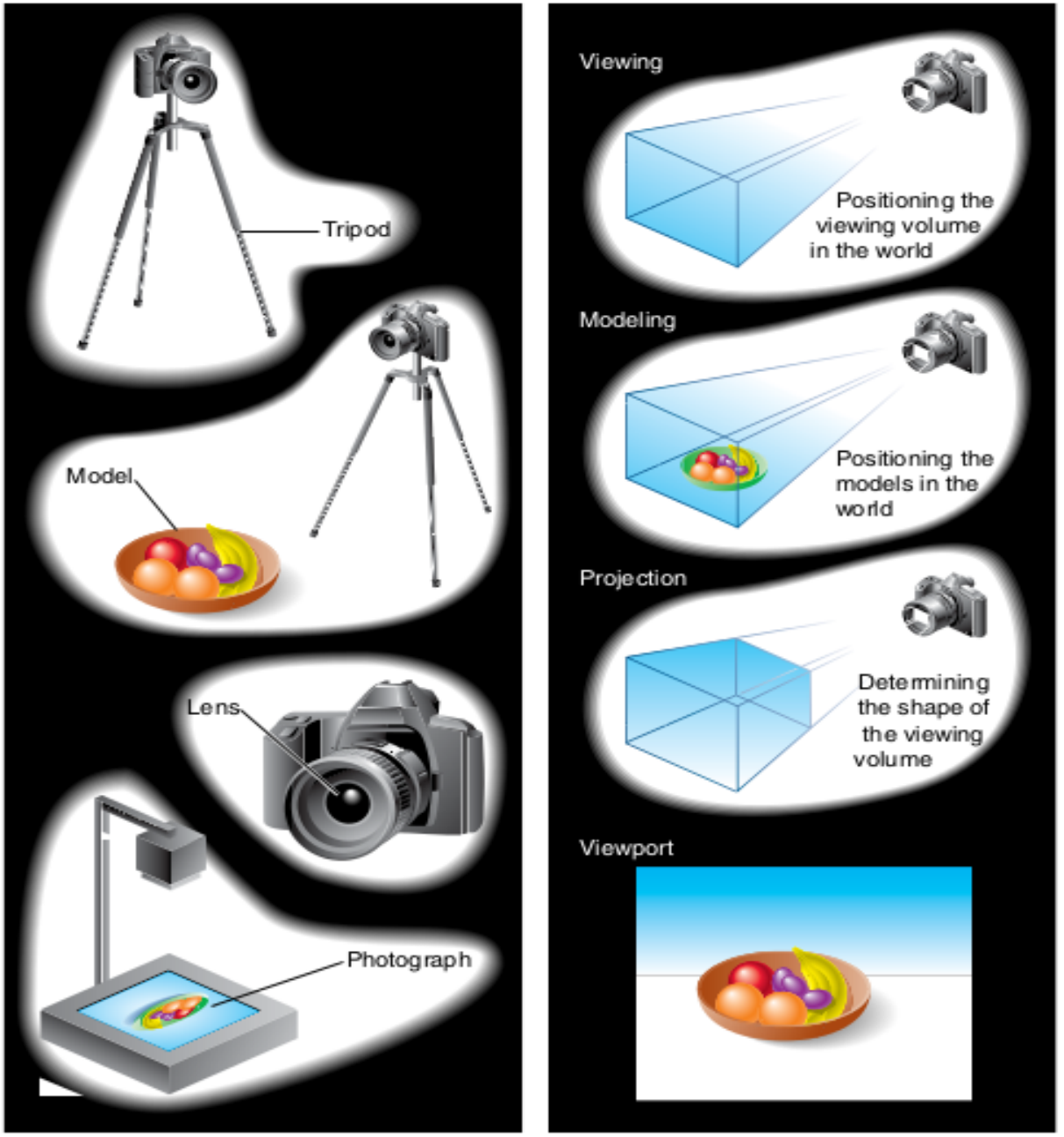
Hendri Karisma
Teknik Informatika
Universitas Komputer Indonesia
2013

Materi

- Overview: The Camera Analogy
- Viewing and Modeling Transformations
- Projection Transformations
- Viewport Transformation
- Manipulating the Matrix Stacks

Overview: The Camera Analogy

- Set up your tripod and point the camera at the scene (viewing Transformation).
- Arrange the scene to be photographed into the desired composition (modeling transformation).
- Choose a camera lens or adjust the zoom (projection transformation).
- Determine how large you want the final photograph to be—for example, you might want it enlarged (viewport transformation).


Tripod
Model
Lens
Photograph
Viewing
Positioning the viewing volume in the world
Modeling
Positioning the models in the world
Projection
Determining the shape of the viewing volume
Viewport

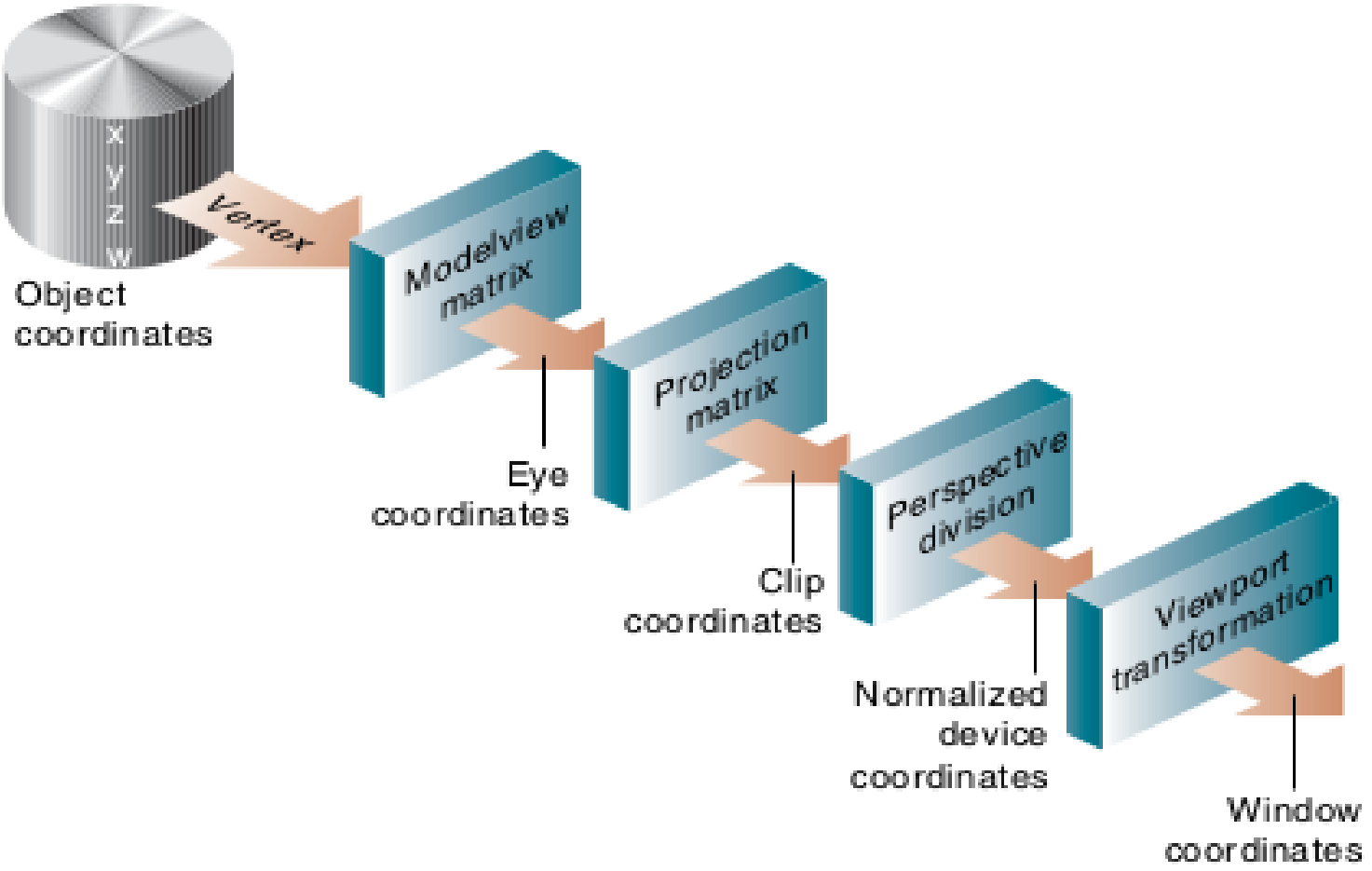


Figure 3-2 Stages of Vertex Transformation

Viewing Transformation

- Recall that the viewing transformation is analogous to positioning and aiming a camera. In this code example, before the viewing transformation Viewing can be specified, the current matrix is set to the identity matrix with glLoadIdentity().

The Viewport Transformation

- Together, the projection transformation and the viewport transformation determine how a scene is mapped onto the computer screen.

General-Purpose Transformation Commands

- glMatrixMode()

  Specifies whether the modelview, projection, or texture matrix will be modified, using the argument GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE for mode.

- GlLoadIdentity()

  To clear the currently modifiable matrix for future transformation commands, as these commands modify the current matrix.

- GlLoadMatrix*(const m)

  To specify explicitly a particular matrix to be loaded as the current matrix, use glLoadMatrix*() or glLoadTransposeMatrix*().

- glLoadTransposeMatrix*()
- glMultMatrix*(),

Viewing and Modeling Transformations

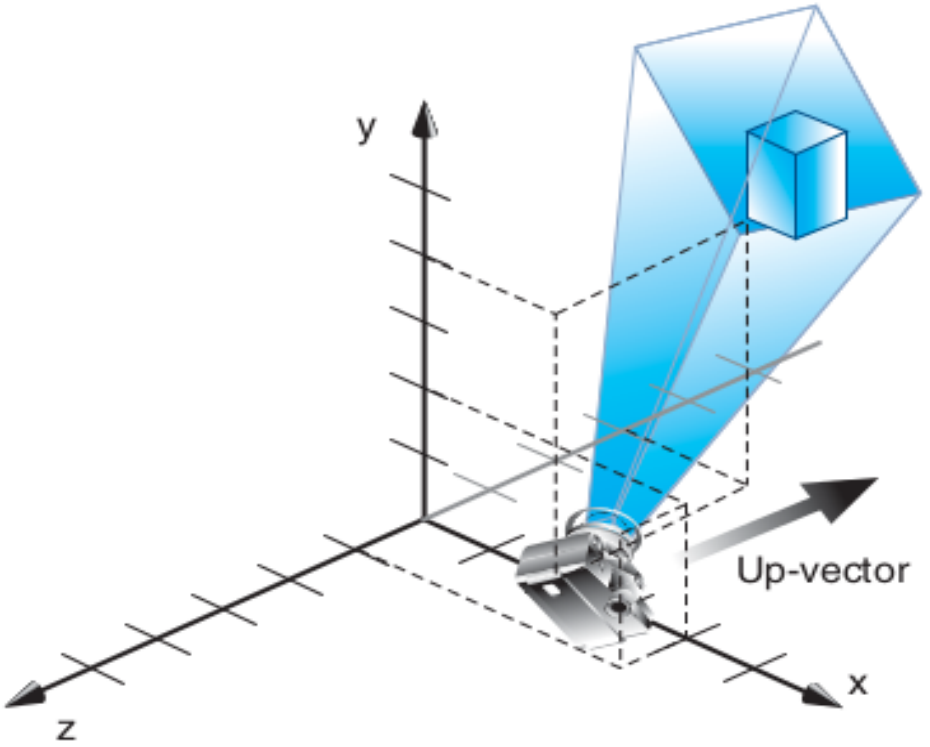
- Viewing and modeling transformations are inextricably related in OpenGL and are, in fact, combined into a single modelview matrix.

Using the gluLookAt() Utility Routine

void **gluLookAt**(GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*,
GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*,
GLdouble *upx*, GLdouble *upy*, GLdouble *upz*);

Defines a viewing matrix and multiplies it to the right of the current matrix. The desired viewpoint is specified by *eyex*, *eyey*, and *eyez*. The *centerx*, *centery*, and *centerz* arguments specify any point along the desired line of sight, but typically they specify some point in the center of the scene being looked at. The *upx*, *upy*, and *upz* arguments indicate which direction is up (that is, the direction from the bottom to the top of the viewing volume).

- gluLookAt(4.0, 2.0, 1.0, 2.0, 4.0, -3.0, 2.0, 2.0, -1.0);

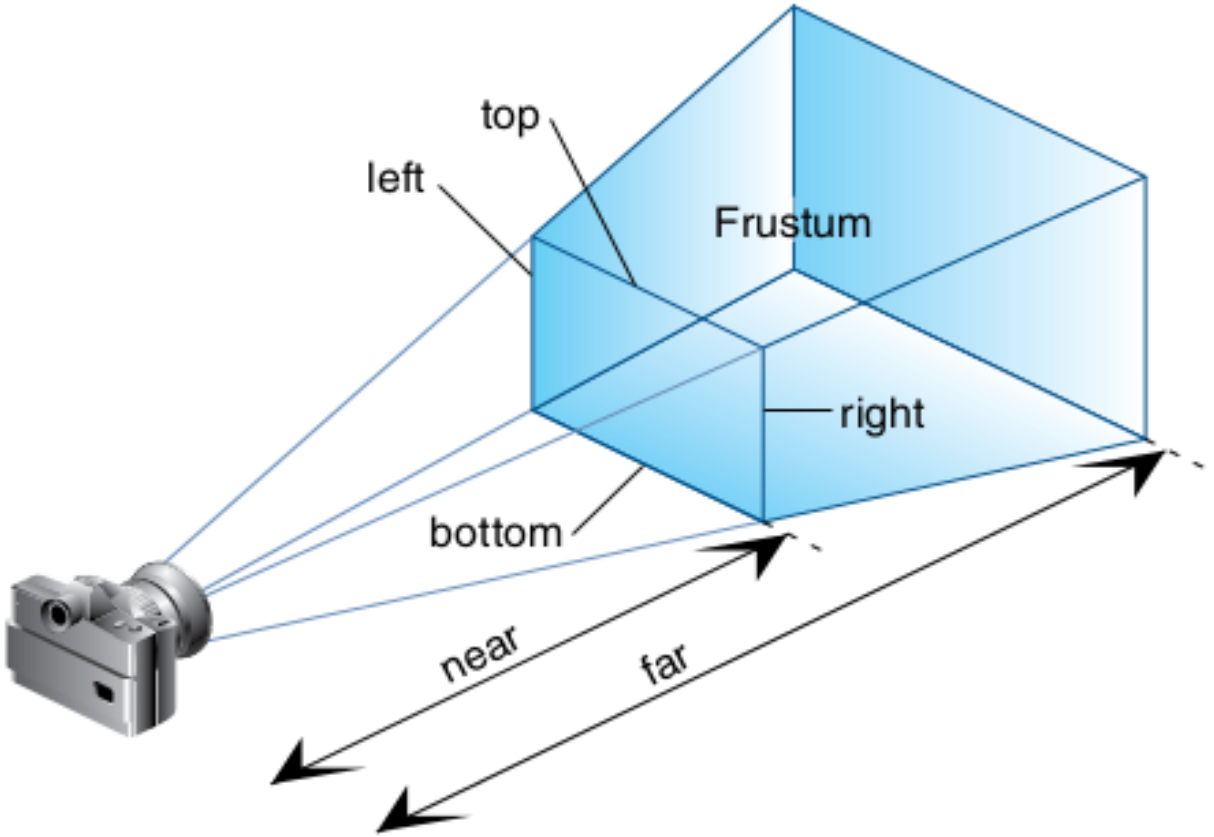


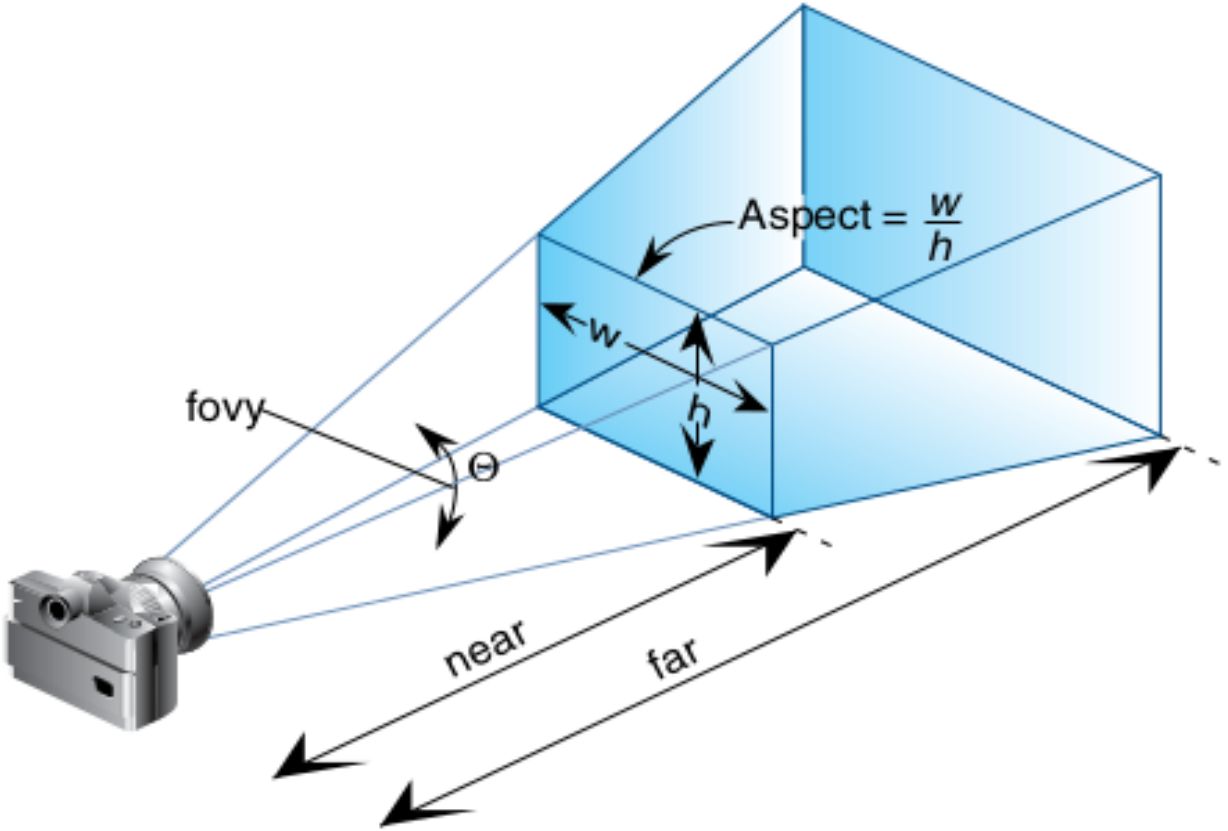
gluFrustum

void **glFrustum**(GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdouble *far*);

Creates a matrix for a perspective-view frustum and multiplies the current matrix by it. The frustum's viewing volume is defined by the parameters: (*left, bottom, –near*) and (*right, top, –near*) specify the (*x, y, z*) coordinates of the lower left and upper right corners, respectively, of the near clipping plane; *near* and *far* give the distances from the viewpoint to the near and far clipping planes. They should always be positive.

gluFrustum




Aspect = w/h
w
h
fovy
Θ
near
far

gluPerspective

void **gluPerspective**(GLdouble *fovy*, GLdouble *aspect*,
GLdouble *near*, GLdouble *far*);

Creates a matrix for a symmetric perspective-view frustum and multiplies the current matrix by it. *fovy* is the angle of the field of view in the *yz*-plane; its value must be in the range [0.0, 180.0]. *aspect* is the aspect ratio of the frustum, its width divided by its height. *near* and *far* values are the distances between the viewpoint and the clipping planes, along the negative *z*-axis. They should always be positive.

gluOrtho

void **glOrtho**(GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdouble *far*);

Creates a matrix for an orthographic parallel viewing volume and multiplies the current matrix by it. (*left, bottom, –near*) and (*right, top, –near*) are points on the near clipping plane that are mapped to the lower left and upper right corners of the viewport window, respectively. (*left, bottom, –far*) and (*right, top, –far*) are points on the far clipping plane that are mapped to the same respective corners of the viewport. Both *near* and *far* may be positive, negative, or even set to zero. However, *near* and *far* should not be the same value.