3D Graphics and API with OpenGL

Human-Centered CAD Lab.

Contents

- 3D Graphics API & OpenGL
- Interactive Computer Graphics
- Example of OpenGL Programming
  - Preparatory
  - Simple code
  - GLUT Functions
- Transformation Matrix
- Matrix Manipulation
- Other Settings

3D Graphics API & OpenGL (1)

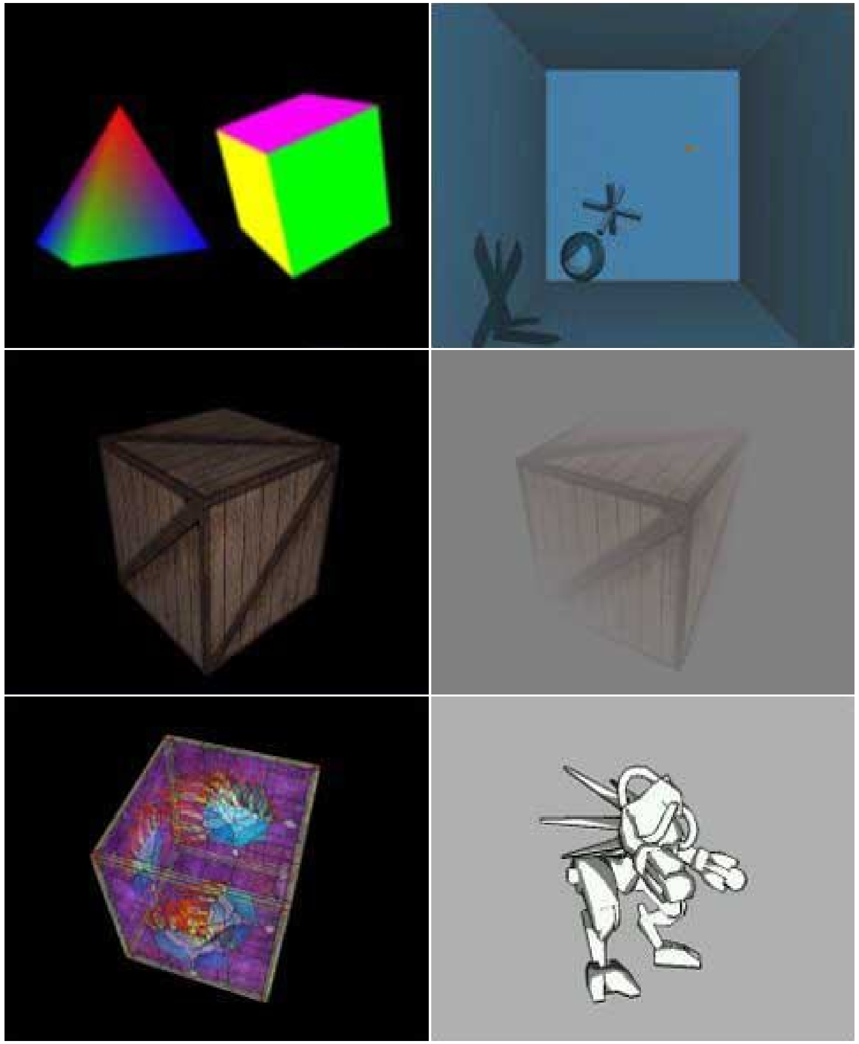
- 3D Graphics = modeling + rendering + animation
  - A model is a computer representation of an object
  - Rendering is the process of creating an image from a model
  - Animation is the process of generating repeated renderings of a scene
- Graphics API(Application Programming Interface)
  - We need interfaces to the graphics acceleration hardware.
  - The most popular 3D modeling technique is polygon or triangular mesh
  - The set of vertices/polygons enters the rendering pipeline
  - Two best-known interfaces are OpenGL and DirectX.

3D Graphics API & OpenGL (2)

- What is OpenGL ?
  - OpenGL is an acronym for Open Graphics Library
    - OpenGL Architecture Review Board (ARB)
    - Version 2.1 released on August 2, 2006
  - Most Widely Adopted Graphics Standard
    - Easy-to-use / Well-documented
    - High Visual Quality and Performance
    - Portable/Reliable/Stable/Scalable
    - Low-level graphics commands
  - Platform-independent graphics API.
    - UNIX, Linux, Windows9X/NT/2000/XP, OS/2, MacOS, BeOS, etc.

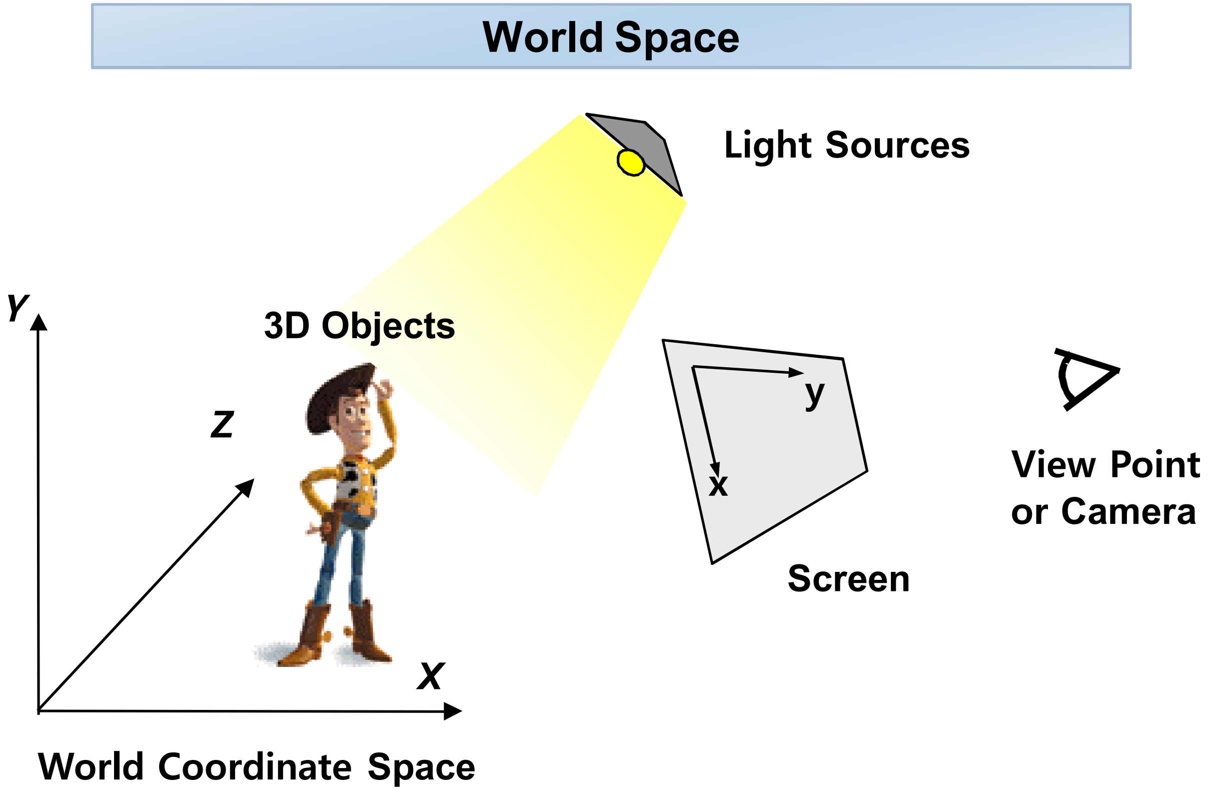
3D Graphics API & OpenGL (3).

- What can we do with OpenGL?
  - Modeling Primitives
  - Drawing Curve/Surfaces
  - Colors and Shading
  - Lights and Shadows
  - Texture mapping
  - Fog / Anti-aliasing
  - Blending / Transparency
  - And… So many things.



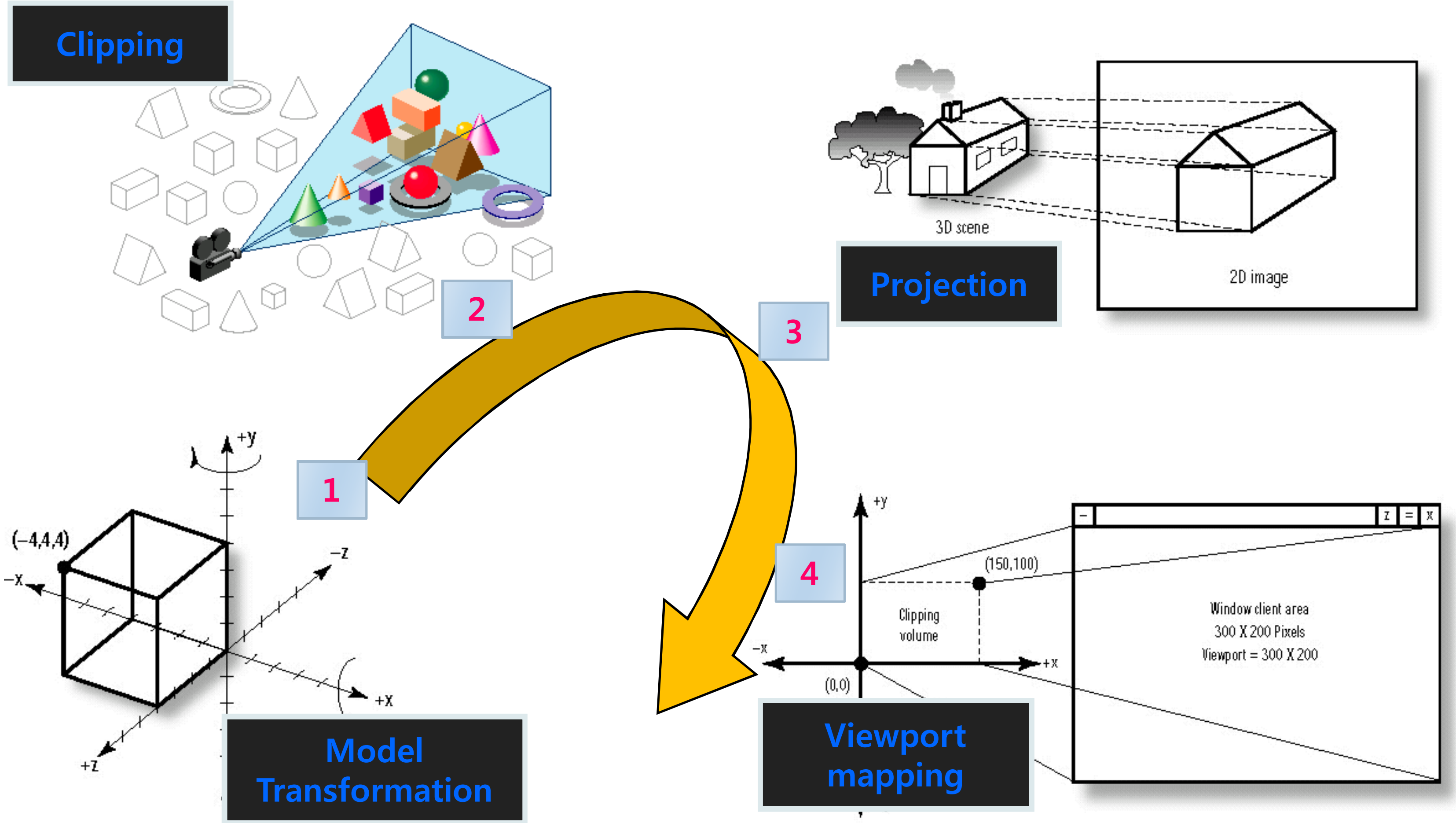
Interactive Computer Graphics (1)

- Problem Specification
  - 3D Objects → 2D Shaded image



Interactive Computer Graphics (2)

- 3D Objects to 2D image



Clipping

2

3

Projection

1

4

Model Transformation

Viewport mapping

Interactive Computer Graphics (3)

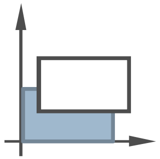
- Transformation Matrix
  - 3D coordinate transformation
  - 3D coordinate → 4D homogeneous coordinate
  - rotate/translate/scale/project is represented by 4x4 matrix
  - All transformation is implemented by multiplying matrices

Some Studies in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

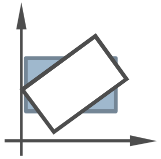
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Interactive Computer Graphics (4)



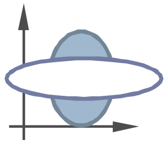
$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation



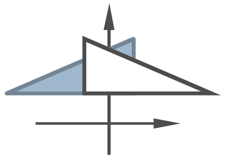
$$\begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation (about z-axis)



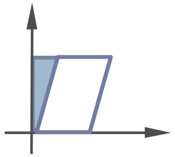
$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling



$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Mirroring (about yz-plane)

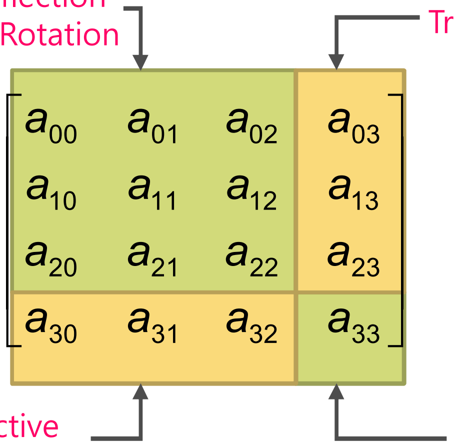


$$\begin{bmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shearing

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} xh \\ yh \\ zh \\ h \end{bmatrix}$$

Homogeneous Coordinates



Scaling / Reflection / Shearing / Rotation

Translation

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Perspective transformation

Global Scaling

4x4 Transformation Matrix

Interactive Computer Graphics (5)

- Vertex transformation for each step in OpenGL

$$\begin{bmatrix} x_O \\ y_O \\ z_O \\ w_O \end{bmatrix} \rightarrow \begin{bmatrix} \text{ModelView} \\ \text{Matrix} \end{bmatrix} \rightarrow \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} \rightarrow \begin{bmatrix} \text{Projection} \\ \text{Matrix} \end{bmatrix} \rightarrow \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \rightarrow \begin{bmatrix} \text{Perspective} \\ \text{Division} \end{bmatrix} \rightarrow \begin{bmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{bmatrix} \rightarrow \cdots$$

Original Vertex

Transformed Eye coordinates

Clip coordinates

Normalized Device coordinates

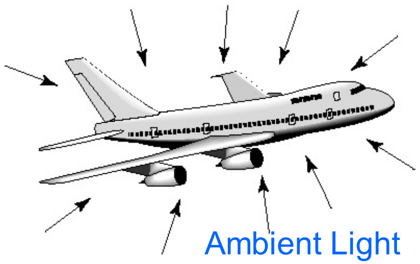
$$\cdots \rightarrow \begin{bmatrix} \text{Viewport} \\ \text{Transformation} \end{bmatrix} \rightarrow$$

Window coordinates

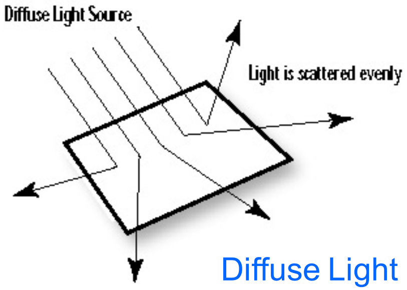
Most geometric transformation → “4x4 Matrix Multiplication”

Interactive Computer Graphics (6)

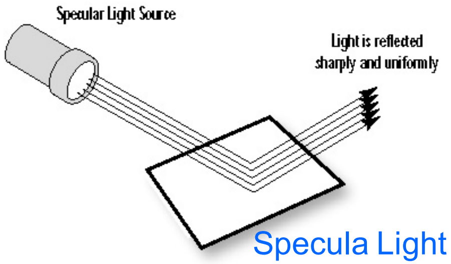
- Lighting
  - Color of point calculated with material property, normal vector, position of light



Ambient Light

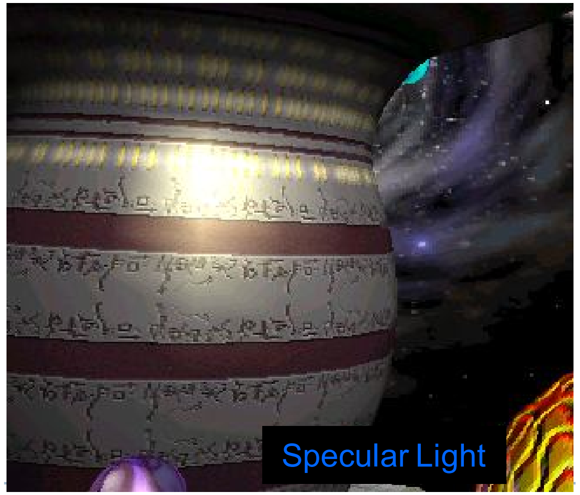


Diffuse Light



Specula Light

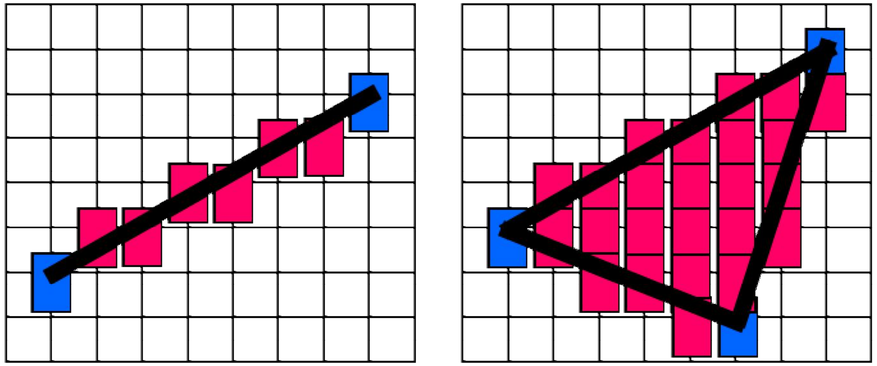
Diffuse Light



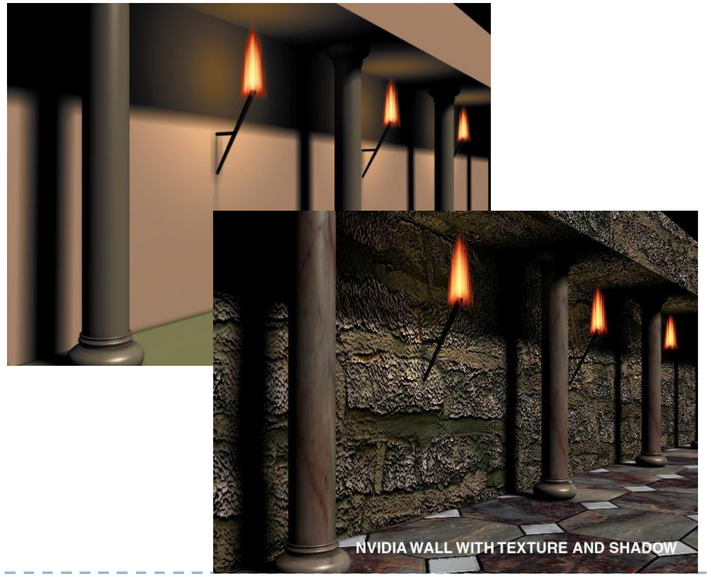
Interactive Computer Graphics (7)

- Rasterization
  - Lighting
  - Scan Converting
  - Hidden Surface Removal
  - Texture Mapping

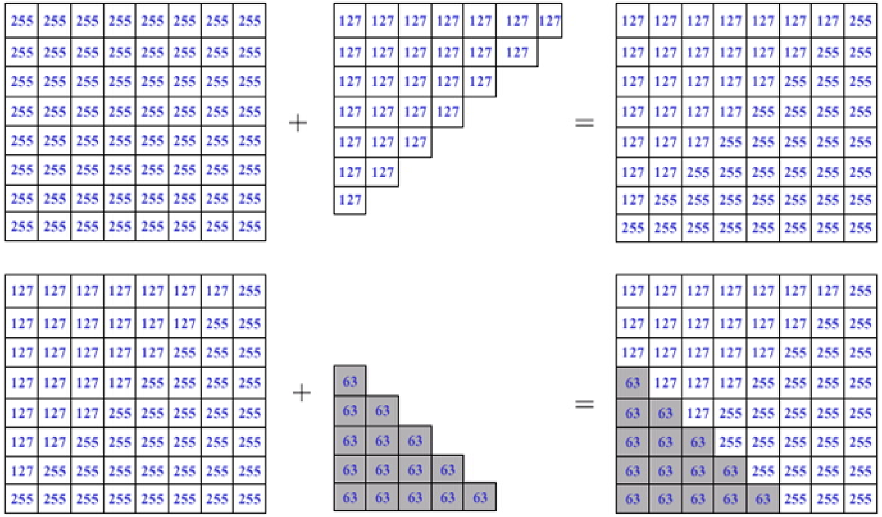
Scan Conversion of a line and a triangle



Benefits of the texture-mapping



Hidden Surface Removal with Z-buffer method



255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

+

127	127	127	127	127	127	127
127	127	127	127	127	127	
127	127	127	127	127		
127	127	127	127			
127	127	127				
127	127					
127						

=

127	127	127	127	127	127	127	255
127	127	127	127	127	127	255	255
127	127	127	127	127	255	255	255
127	127	127	127	255	255	255	255
127	127	127	255	255	255	255	255
127	127	255	255	255	255	255	255
127	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

127	127	127	127	127	127	127	255
127	127	127	127	127	127	255	255
127	127	127	127	127	255	255	255
127	127	127	127	255	255	255	255
127	127	127	255	255	255	255	255
127	127	255	255	255	255	255	255
127	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

+

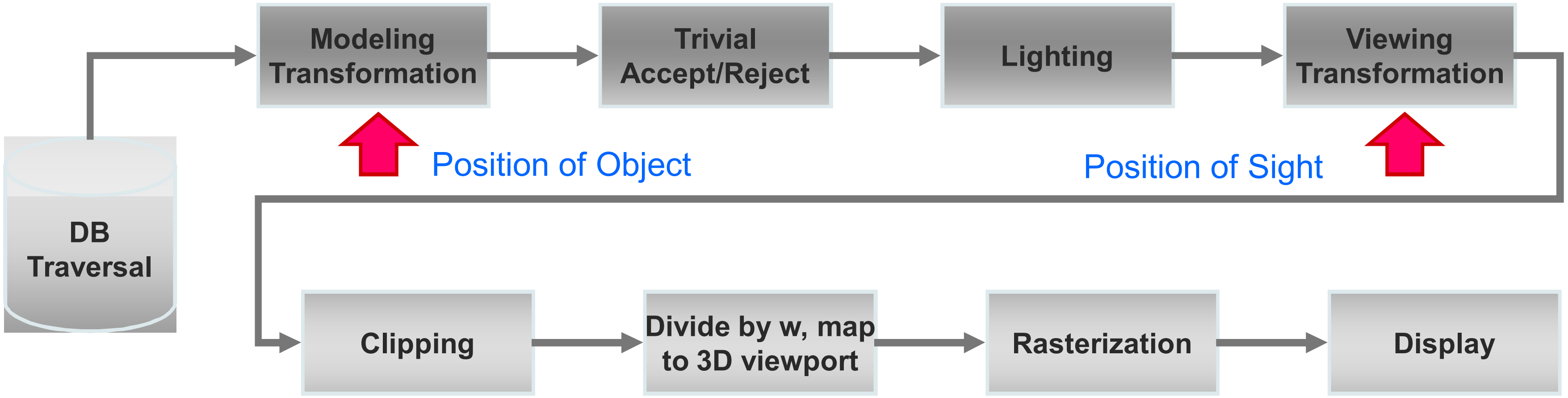
63				
63	63			
63	63	63		
63	63	63	63	
63	63	63	63	63

=

127	127	127	127	127	127	127	255
127	127	127	127	127	127	255	255
127	127	127	127	127	255	255	255
63	127	127	127	255	255	255	255
63	63	127	255	255	255	255	255
63	63	63	255	255	255	255	255
63	63	63	63	255	255	255	255
63	63	63	63	63	255	255	255

Interactive Computer Graphics (8).

- Standard 3D Graphics Pipeline
  - Model Coord. → World Coord. → View Coord.→ Lighting → Projection → Frustum Culling → Normalized Device Coord. → ViewPort → Rasterization
- OpenGL is..
  - Parallel process of geometric primitives and image data
  - State Machine / Z-buffer & Gouraud Shading

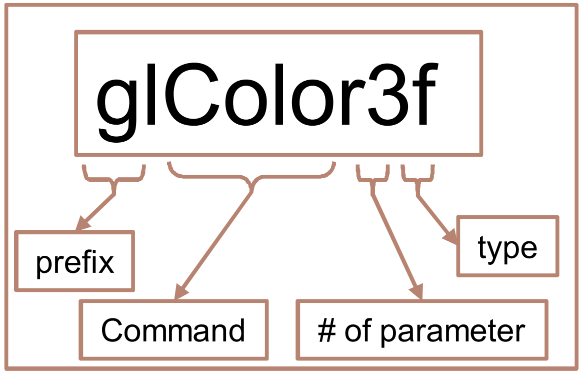


Preparatory (1)

- Needs for OpenGL programming
  - Device driver for Hardware acceleration of OpenGL
    - nVidia / ATI graphic card drivers
  - Development Toolkit (IDE)
    - Microsoft Visual Studio6, .NET 2003, .NET2005
  - OpenGL Programming Environment
    - header(.h) / library(.lib)
      - □ Header files : gl.h, glu.h, glaux.h, glut.h
      - □ Library files : opengl32.lib, glu32.lib, glut32.lib (by Microsoft)
        opengl.lib, glu.lib, glut.lib (by SGI)
  - Microsoft natively supports OpenGL
    - From Windows 98/NT4.0, but NOT include GLUT

Preparatory (2)

- OpenGL Command Syntax
  - About 130 functions.
  - All of functions has a prefix gl
    - Ex) glTranslate3f()
  - All of variables has a prefix GL_
    - Ex) GL_LIGHTING



<Example>
glColor3b
glColor3i
glColor3f
glColor3d
…
glColor4f
glColor4fv
…

Suffix	C++ type	OpenGL Type Definition
b	signed char	GLbyte
s	short	GLshort
i	long	GLint, GLsizei
f	float	GLfloat, GLclampf
d	double	GLdouble, GLclampd
ub	unsigned char	GLubyte, GLboolean
us	unsigned short	GLushort
ui	unsigned long	GLuint, GLenum, GLbitfield

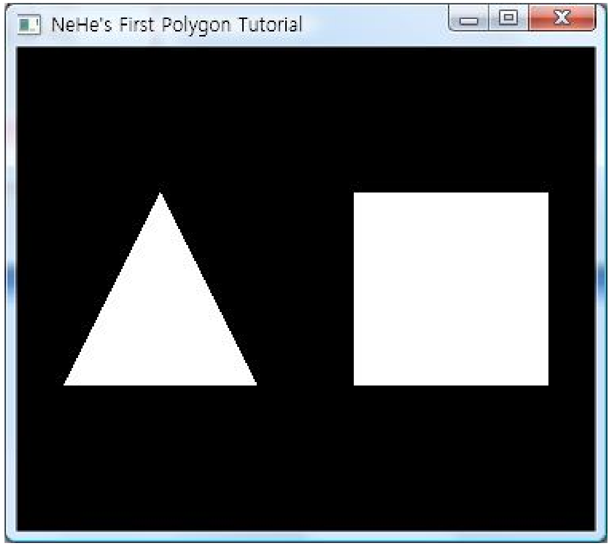
Preparatory (3).

- OpenGL related Libraries
  - OpenGL Utility library (GLU)
    - Prefix glu `gluLookAt();`
  - OpenGL Programming Guide Auxiliary library (AUX)
    - Prefix aux `auxSolidBox()`
  - OpenGL Utility Toolkits (GLUT)
    - Prefix glut `glutSolidCube()`
    - by Mark J. Kilgard, Silicon Graphics, Inc.
  - High-level Graphics Library
    - Open Inventor
    - OpenGL Performer, OpenGL Optimizer .etc

Simple code (1)

- Chunk of OpenGL code

```
#include <whateverYouNeed.h>
main()
{
            InitializeAWindowPlease();
            glClearColor (1.0, 1.0, 1.0, 1.0);
            glClear (GL_COLOR_BUFFER_BIT);
            glColor3f (0.0, 0.0, 0.0);
            glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);

            glBegin(GL_POLYGON);
                        glVertex3f (0.25, 0.25, 0.0);
                        glVertex3f (0.75, 0.25, 0.0);
                        glVertex3f (0.75, 0.75, 0.0);
                        glVertex3f (0.25, 0.75, 0.0);
            glEnd();
            glFlush();
            UpdateTheWindowAndCheckForEvents();
}
```



Simple code (2)

- Sample code using GLAUX library

```
#include "windows.h"
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

void myinit (void)
{
     glClearColor (1.0, 1.0, 1.0, 1.0);
}

void CALLBACK display(void)
{
     glClear (GL_COLOR_BUFFER_BIT);

     glColor3f (0.0, 0.0, 0.0);
     glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
     glEnd ();

     glFlush ();
}
```

```
void CALLBACK myReshape(GLsizei w, GLsizei h)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
        glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
        auxInitDisplayMode (AUX_SINGLE | AUX_RGB);
        auxInitPosition (0, 0, 500, 500);
        auxInitWindow ("Smooth Shading");
        myinit();
        auxReshapeFunc (myReshape);
        auxMainLoop(display);
        return 0;
}
```

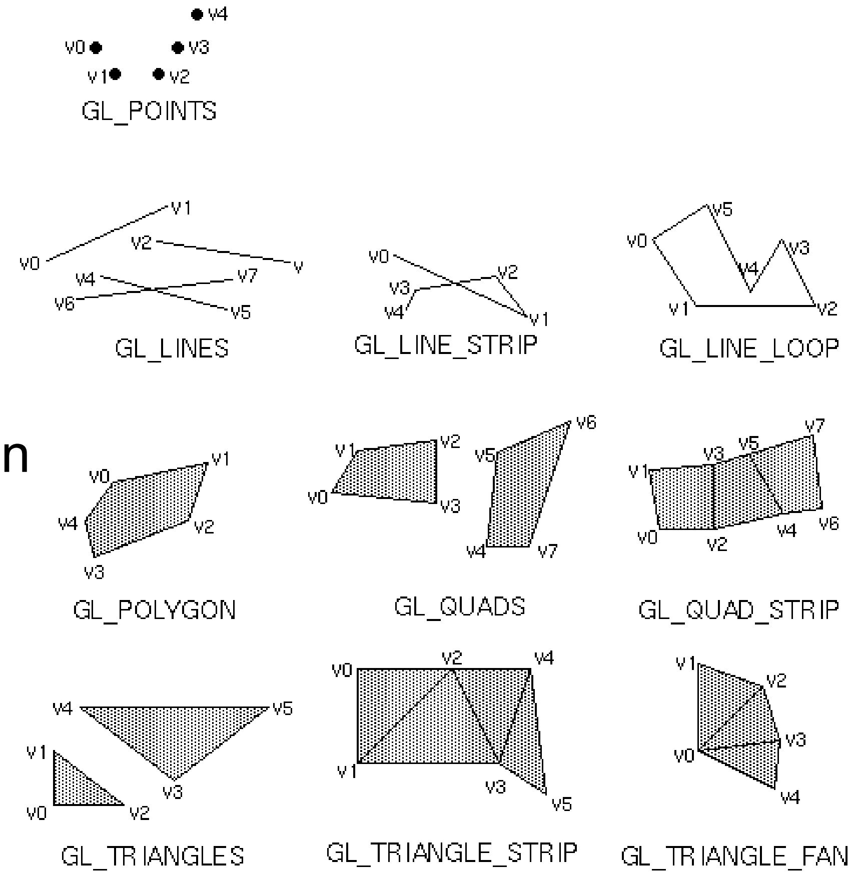
Simple code (3).

- Drawing Objects
  - Create vertices

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.14159265);
glVertex4f(2.3f, 1.0f, -2.2f, 2.0f);
```

  - Create primitives
    - Point, Line, Triangle, Polygon

```
glBegin(GL_POLYGON);
            glVertex2f(0.0, 0.0);
            glVertex2f(0.0, 3.0);
            glVertex2f(4.0, 3.0);
            glVertex2f(6.0, 1.5);
            glVertex2f(4.0, 0.0);
glEnd();
```



GLUT Functions (1)

Initializing and Exiting a Window

-void glutInit(int *argcp, char **argv);
-void glutInitDisplayMode(unsigned int mode);
-void glutInitWindowSize(int width, int height);
-void glutInitWindowPosition(int x, int y);
-void glutDestroyWindow(int win);

Loading the Color Map

-void glutSetColor(int cell, GLfloat red, GLfloat green, GLfloat blue);
-GLfloat glutGetColor(int cell, int component);
-void glutSwapBuffers(void);
-void glutStrokeCharacter(void *font, int character);

Managing Background Process

-void glutIdleFunc(void (*func)(void));

Running the Program

-void glutMainLoop(void);

GLUT Functions (2).

Initializing and Drawing Three-Dimensional Objects

-void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
-void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
-void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
-void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
-void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
-void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
-void glutSolidCube(GLdouble size);
-void glutSolidDodecahedron(void);
-void glutSolidOctahedron(void);
-void glutSolidTetrahedron(void);
-void glutSolidIcosahedron(void);
-void glutSolidTeapot(GLdouble size);
-void glutWireCube(GLdouble size);
-void glutWireDodecahedron(void);
-void glutWireOctahedron(void);
-void glutWireTetrahedron(void);
-void glutWireIcosahedron(void);
-void glutWireTeapot(GLdouble size);

Handling Window and Input Events

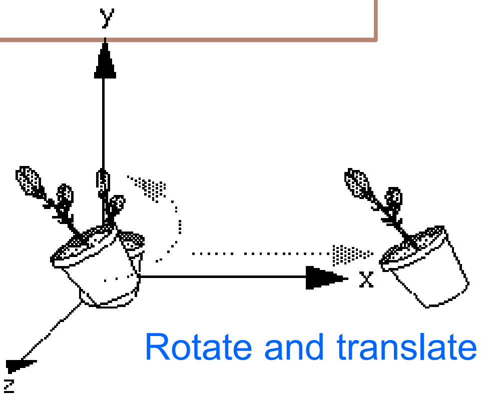
-void glutReshapeFunc(void (*func)(int width, int height));
-void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
-void glutMouseFunc(void (*func)(int button, int state, int x, int y));

Transformation Matrix (1)

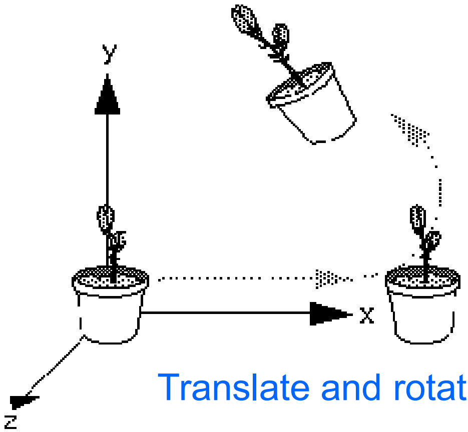
- General Transform Command
  - glMultiMatrix*() function - 4x4 matrix multiplication
  - The order of transformation is important

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);
glMultMatrixf(M);
glMultMatrixf(L);
glBegin(GL_POINTS);
glVertex3f(v);
glEnd();
```

Transformation result = N (M (L · v)
= N ·M ·L ·v



Rotate and translate



Translate and rotate

Transformation Matrix (2)

- Matrix Mode
  - ModelView matrix mode [GL_MODELVIEW]
    - Transform object position (change local coordinate)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

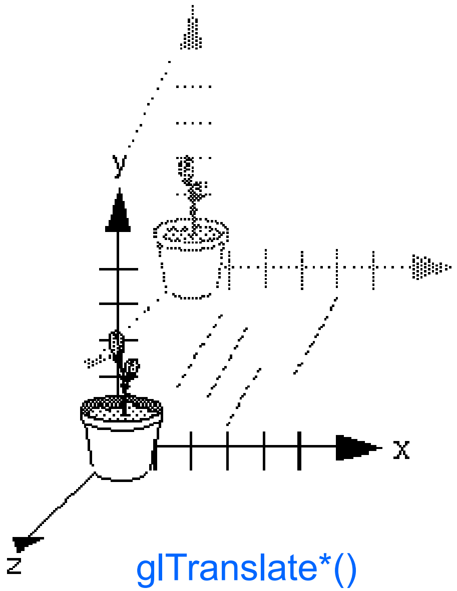
  - Projection matrix mode [GL_PROJECTION]
    - Matrix for 2D projection
    - Define the viewing volume

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

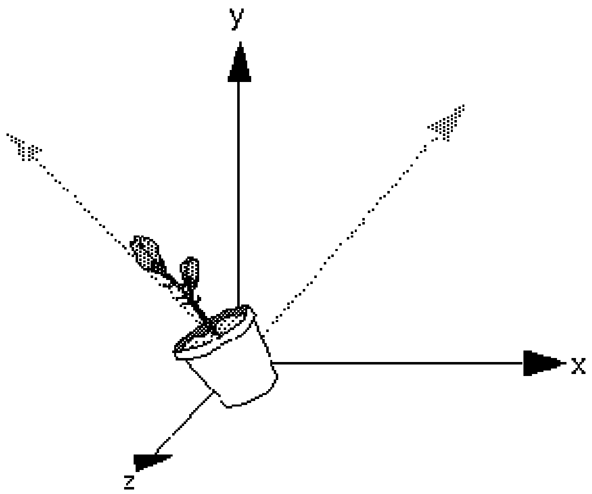
Transformation Matrix (3)

- Modeling Transform [GL_MODELVIEW]
  - 3 routine for modeling transform in OpenGL

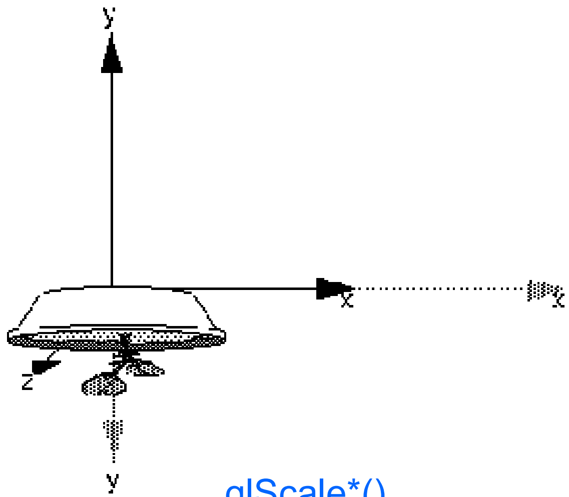
glTranslate*(), glRotate*(), glScale*()



glTranslate*()



glRotate*()

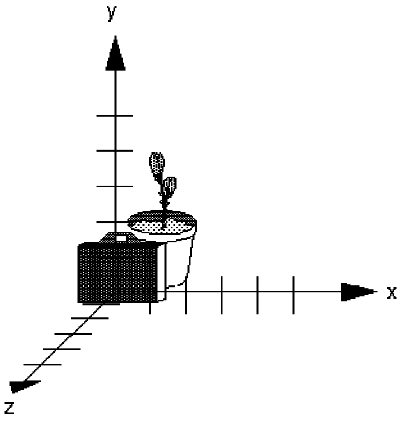


glScale*()

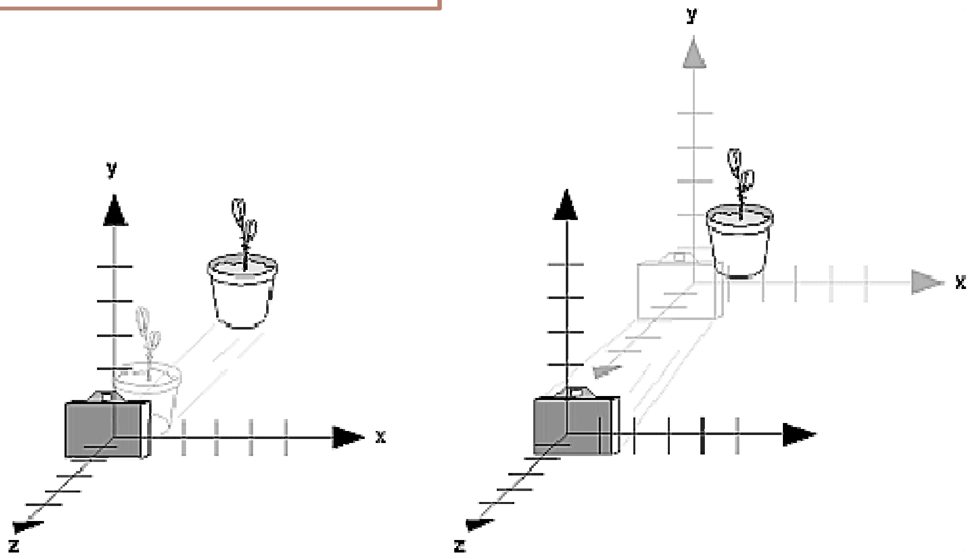
Transformation Matrix (4)

- Viewing Transform [GL_MODELVIEW]
  - 2 routine for viewing transform in OpenGL

glTranslate*(), glRotate*()



Initial state of
viewing vs. modeling coord.

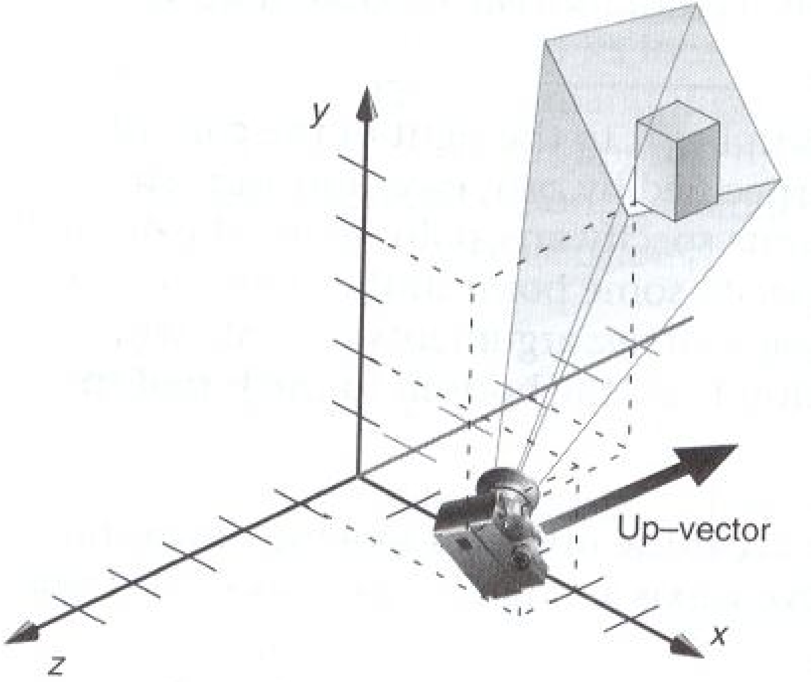


glTranslatef(0.0, 0.0, -5.0);

Transformation Matrix (5)

- Viewing Transform [GL_MODELVIEW]
  - gluFunction to define the viewing transform easily

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

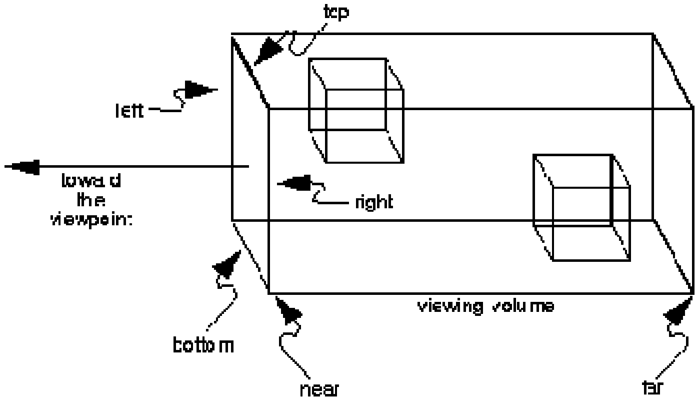


Transformation Matrix (6)

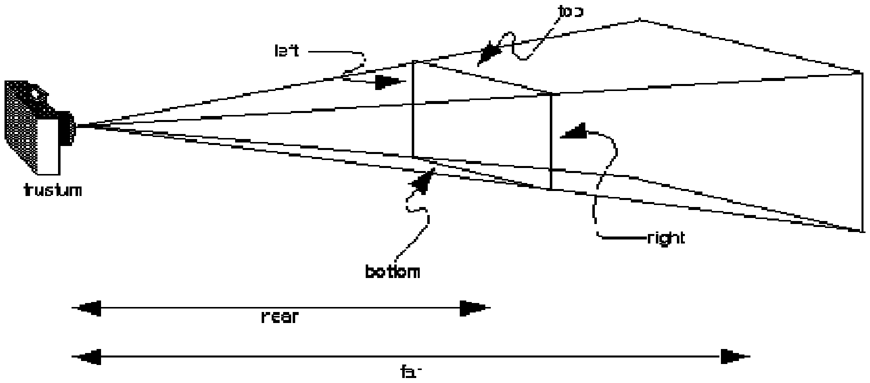
- Projection Transform [GL_PROJECTION]
  - Viewing volume is determined by view coordinate
    - So must be called in Projection Matrix mode

```
void glOrtho( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar )
```

```
void glFrustum( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar)
```



orthogonal projection using glOrtho()

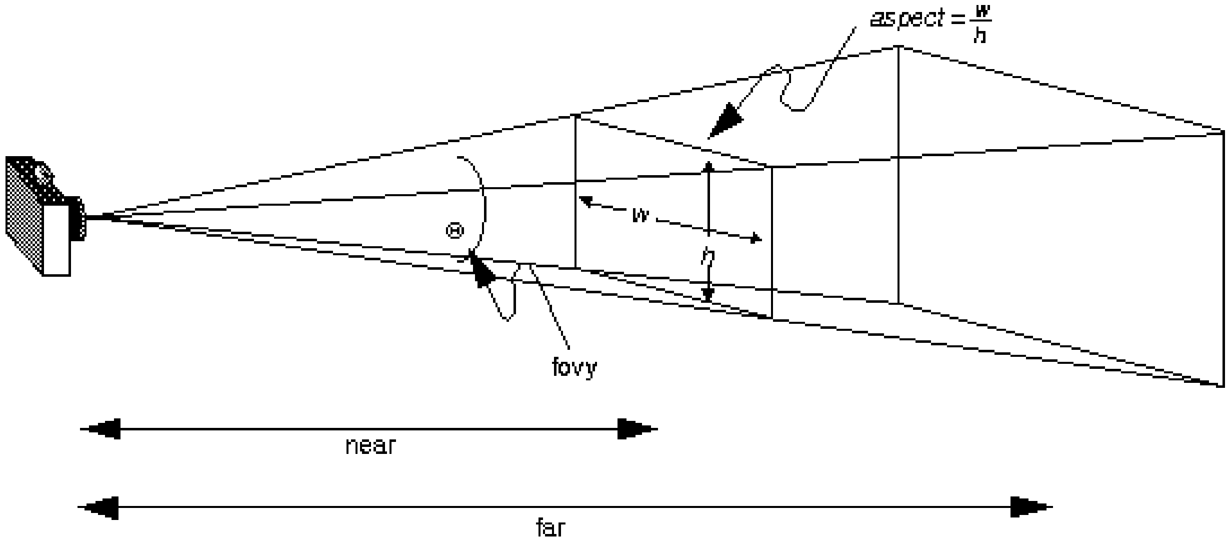


perspective projection using glFrustum()

Transformation Matrix (7)

- Projection Transform [GL_PROJECTION]
  - gluFunction to define viewing volume easily

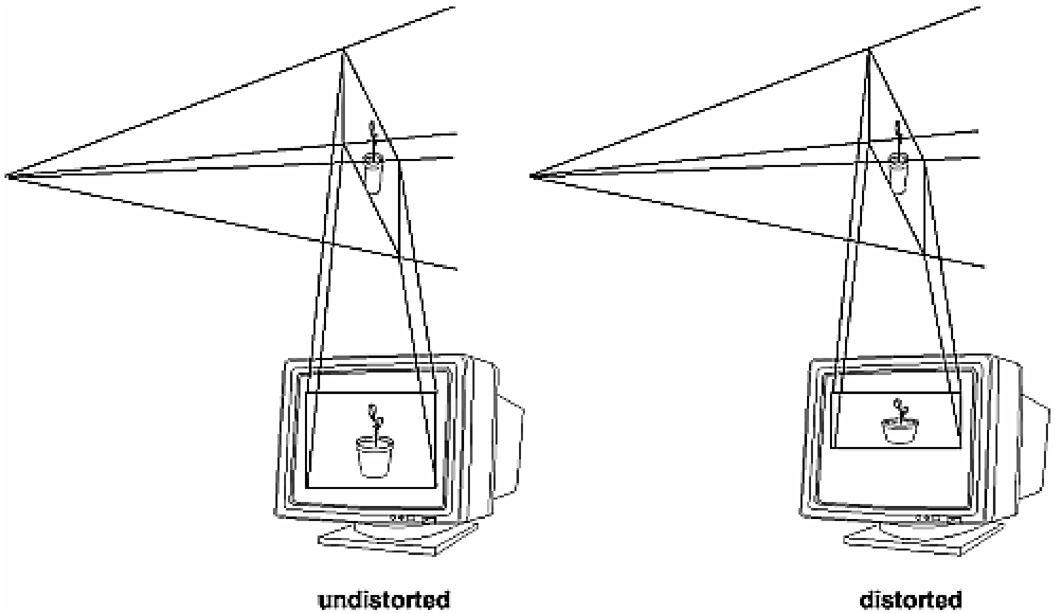
```
void gluPerspective( GLdouble fovy, GLdouble aspect, GLdouble znear, GLdouble zfar)
```



Perspective viewing volume using gluPerspective()

Transformation Matrix (8).

- Viewport Transform
  - Mapping the projected image to the 2D window
  - May expect distortion from different aspect ratio
  - Can use multiple viewport



Distortion from difference between the aspect ratios of viewing volume and window

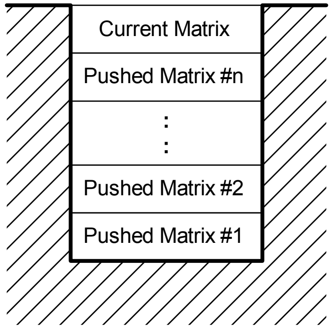
** MUST change the aspect ratio of viewing volume to fit with that of window

Matrix Manipulation (1)

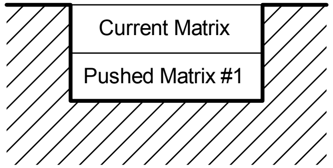
- Push/Pop Matrix
  - To draw several objects with hierarchy, we must save transformation matrix of previous step
    - assembly #n: A···N · M
    - sub-assembly #ni: A···N · M · L
    - assembly #n+1: A···N · M

```
glPushMatrix();
glMultiMatrix(L);
DrawSubAssy(ni);
glPopMatrix();
```

- Matrix Stack of ModelView / Projection Matrix



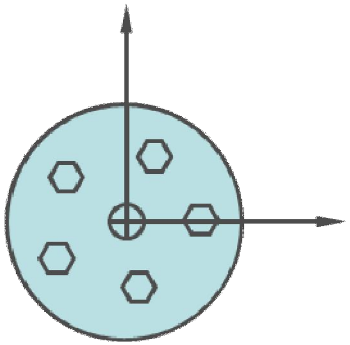
Modelview
Matrix Stack
(32, 4*4 Matrix)

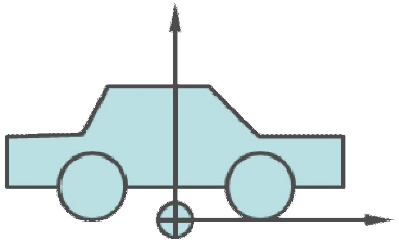


Projection
Matrix Stack
(2, 4*4 Matrix)

Matrix Manipulation (2).

- Example code of Push/Pop Matrix





```
void draw_wheel_and_bolts( )
{
        long I;
        draw_wheel( );
        for( i = 0; i < 5; i++ ) {
                glPushMatrix( );
                    glRotatef( 72.0*i 0.0, 0.0, 1.0 );
                    glTranslatef( 3.0, 0.0, 0.0 );
                    draw_bolt( );
                glPopMatrix( );
        }
}
void draw_body_and_wheel_and_bolts ( )
{
        draw_car_body( );
        glPushMatrix( );
                glTranslatef( 40, 0, 30 );          // move to first wheel's
                draw_wheel_and_bolts( );
        glPopMatrix( );
        glPushMatrix( );
                glTranslatef( 40, 0, -30 );         // move to second wheel's
                draw_wheel_and_bolts( );
        glPopMatrix( );
        .........................                   // move to last wheels'
}
```

Other Settings (1)

- Color
  - RGBA mode
    - Set red, green, blue, alpha values as float (0.0 ~ 1.0)
      - □ Ex) 1.0, 0.0, 0.0 → red , 0.5, 0.5, 0.5 → gray
    - Using glColor*() function
      - □ Ex) glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
  - Color Indexed mode
    - Set index of color as color-lookup-table
    - Using glIndex*() function
    - Can specify your own color-table
      - □ Ex) auxSetOneColor(), glutSetOnColor()
  - RGBA mode is widely used.

Other Settings (2)

- Lighting
  - Determine the color of the object at final 2D raster graphic
  - Using glLight*() function
  - GL_AMBIENT / GL_DIFFUSE / GL_SPECULAR

w=0
Directional light
w=1
Positional light

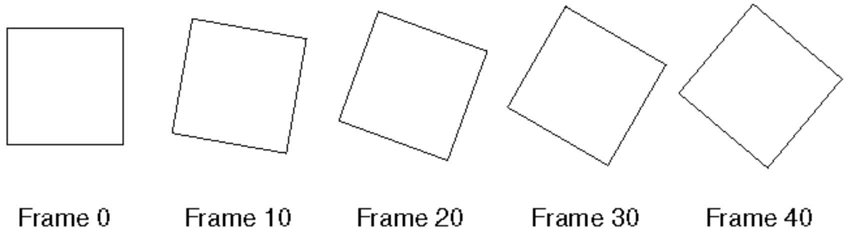
```
GLfloat light_ambient[]     = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[]     = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[]    = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[]    = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

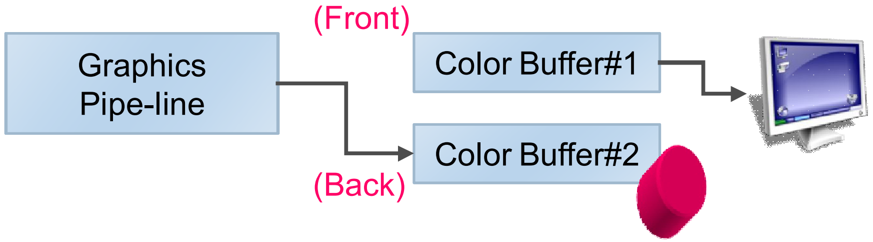
```
GLfloat light1_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat light1_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_position[] = { -2.0, 2.0, 1.0, 1.0 };
GLfloat spot_direction[] = { -1.0, -1.0, 0.0 };

glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);
glLightfv(GL_LIGHT1, GL_POSITION, light1_position);

glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.5);
glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.2);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 45.0);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spot_direction);
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 2.0);

glEnable(GL_LIGHT1);
```

Other Settings (3)

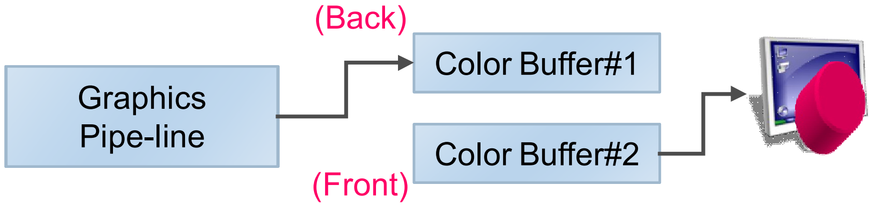
- Double buffering to avoid flickering

```
void glClearColor( GLclampf red, GLclampf green,
                                   GLclampf blue, GLclampf alpha )
        // define default color buffer in RGBA mode
void glClearIndex( GLfloat index )
        // define default color buffer in color-index mode
void glClearDepth( GLclampd depth )
        // depth 0.0 ~ 1.0
void glClearStencil( GLint s )
void glClearAccum(GLfloat red, GLfloat green,
                                    GLfloat blue, GLfloat alpha )
void glClear( GLbitfield mask )
        // initialize frame buffer as default
        // mask      GL_COLOR_BUFFER_BIT,
                     GL_DEPTH_BUFFER_BIT,
                     GL_STENCIL_BUFFER_BIT,
                     GL_ACCUM_BUFFER_BIT
void glDrawBuffer( GLenum mode )
        // define color buffer to draw
        // default   single buffer mode : GL_FRONT
                     double buffer mode : GL_BACK
void glutSwapBuffers();
        //MUST SwapBuffer after draw an object
                              in double-buffering mode
```





glutSwapBuffers();



Other Settings (4).

- Sample code using double-buffering

```
#include "windows.h"
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

void myinit (void)
{
     glClearColor (1.0, 1.0, 1.0, 1.0);
}

void CALLBACK display(void)
{
     glClear (GL_COLOR_BUFFER_BIT);

     glColor3f (0.0, 0.0, 0.0);
     glBegin(GL_POLYGON);
         glVertex3f (0.25, 0.25, 0.0);
         glVertex3f (0.75, 0.25, 0.0);
         glVertex3f (0.75, 0.75, 0.0);
         glVertex3f (0.25, 0.75, 0.0);
     glEnd ();
     glFlush ();
     auxSwapBuffers();
}
```

```
void CALLBACK myReshape(GLsizei w, GLsizei h)
{
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
       glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
       auxInitDisplayMode (AUX_DOUBLE | AUX_RGB);
       auxInitPosition (0, 0, 500, 500);
       auxInitWindow ("Smooth Shading");
       myinit();
       auxReshapeFunc (myReshape);
       auxMainLoop(display);
       return 0;
}
```

** Compare with 18 Page

Useful Site for OpenGL

- OpenGL Manual Page
  - http://www.opengl.org/documentation/specs/man_pages/hardcopy/GL/html/
- OpenGL Utility Toolkit (GLUT) API
  - http://www.opengl.org/documentation/specs/glut/spec3/spec3.html
- OpenGL FAQ and Troubleshooting Guide
  - http://www.opengl.org/resources/faq/technical/
- Forum (Tutorial & Sourcecode)
  - http://nehe.gamedev.net/
  - http://www.codeguru.com/