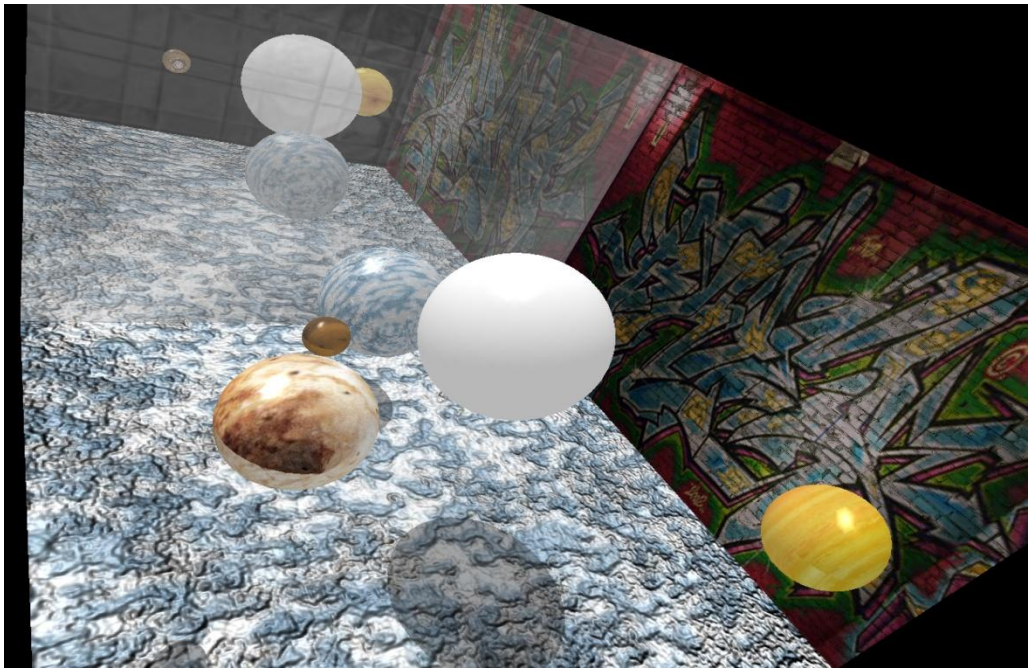
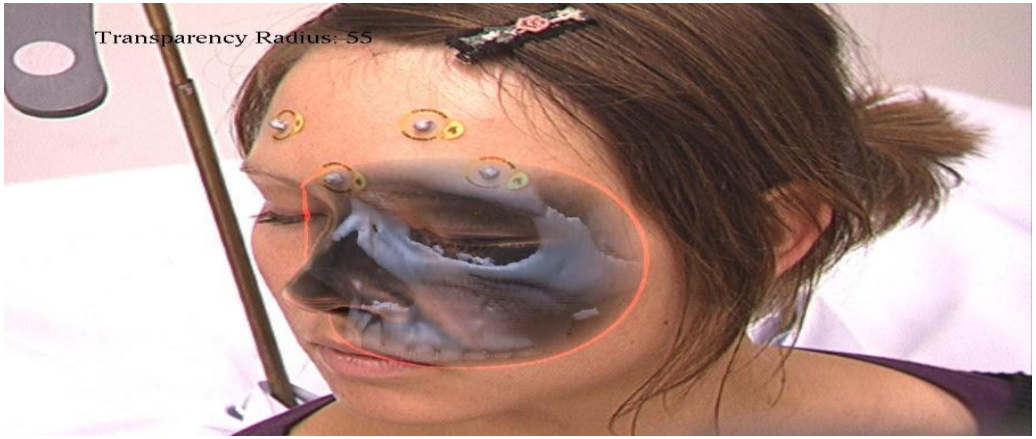
Introduction to OpenGL

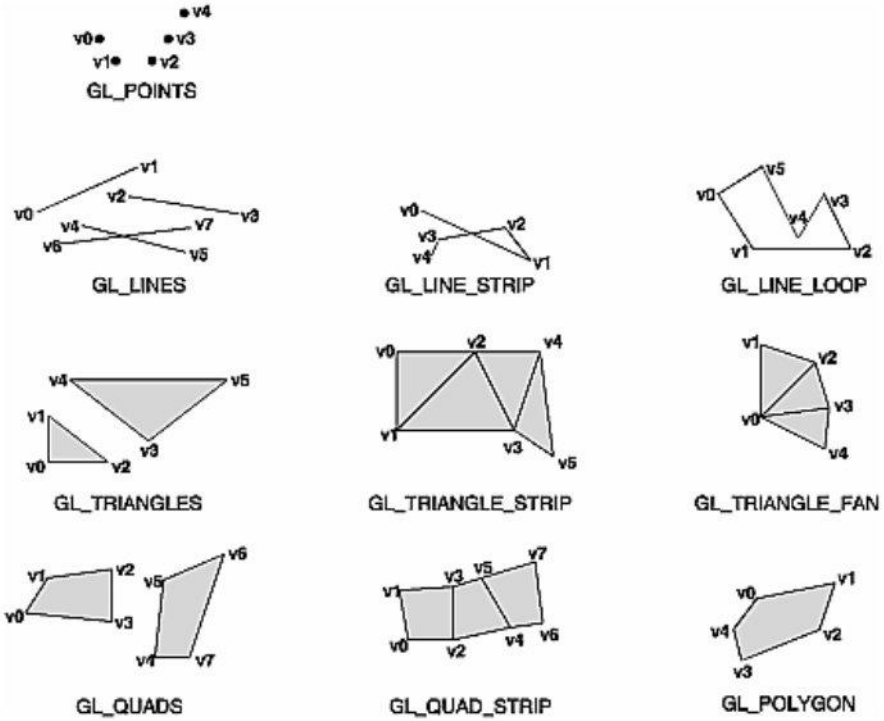
Transformations, Viewing and Lighting

Ali Bigdelou




Transparency Radius: 55

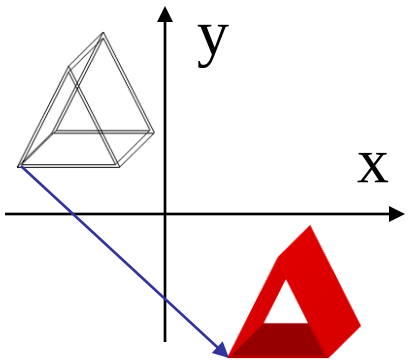
Modeling – From Points to Polygonal Objects



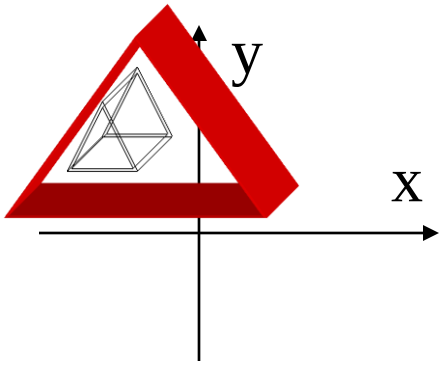
- Vertices (points) are positioned in the virtual 3D scene
- Connect points to form polygons
- Complex objects are composed out of many polygons

Manipulating points (vectors) in 3D space

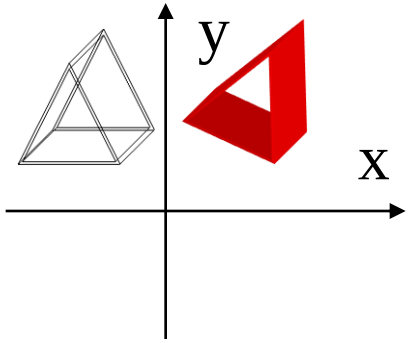
Three important transformations



Translation



Scaling



Rotation

3D Translation & Rotation in Cartesian Coordinates

3D Point

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Translation

$$\vec{p}' = \vec{p} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Rotation around z-axis

$$\vec{p}' = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \vec{p}$$

Performing rotation and traslation

$$\vec{p}' = M_{3x3}\,\vec{p} + t$$

Homogenous Coordinates

- Homogeneous coordinates allow translations to be performed by matrix mutiplication
- The column space is extended with an additional dimension
- Translations can now be represented with a matrix

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Translations matrix

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

Translation

$$\vec{p}' = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \vec{p}$$

Homogenous Coordinates

Rotation matrix

$$\begin{bmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & \vec{0} \\ \vec{0} & 1 \end{bmatrix}$$

Rotation

$$\vec{p} \rightarrow \begin{bmatrix} R & \vec{0} \\ \vec{0} & 1 \end{bmatrix} \vec{p}$$

From $\vec{p}' = M_{3x3}\vec{p} + t$ to $\vec{p}' = M_{4x4}\vec{p}$

Example: 3D Euclidean Transformation

First rotation around the z-axis then translation
(Notice inverse interpretation of multiplication)

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & t_x \\ \sin\alpha & \cos\alpha & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Scaling

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Euler-Angle for Rotation $R=R_zR_yR_x$

Rotation ψ around the z-axis:

$$R_z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation θ around the y-axis:

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

Rotation ϕ around the x-axis:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

OpenGL (Open Graphics Library) since 1992

- API Specification
- Platform-, Programming Language-, Hardware- Independent access to accelerated Graphics-Hardware (Graphics-Cards, GPUs)
- Developed by Silicon Graphics, implemented mostly through graphics drivers
- OpenGL is a state-machine
- Current Version 4.1 (July 26, 2010)
- Extensions add additional functionality to OpenGL
  - Developed by hardware manufacturers
  - Often adopted by other manufacturers
  - Accessibly through libraries like GLEW (OpenGL Extension Wrangler)
  - New OpenGL releases include some extensions into the specification
- Usage: Visualization, Gaming and Augmented Reality
- Websites http://www.opengl.org, http://glew.sourceforge.net/

OpenGL API– Function Naming Convection

<Library prefix><Main command><Optional argument count> <Optional argument type>

Example: glColor3f

gl : Prefix OpenGL function

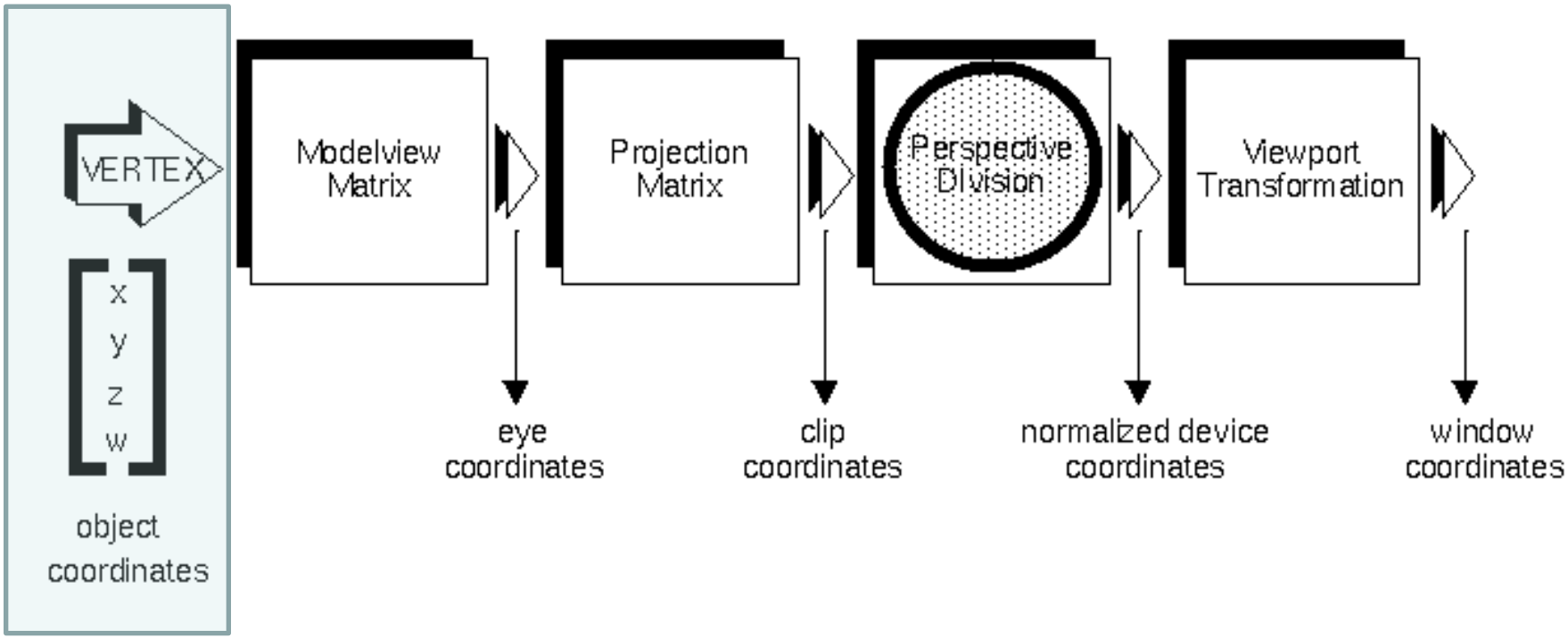
Color : Main Command

3 : number of arguments

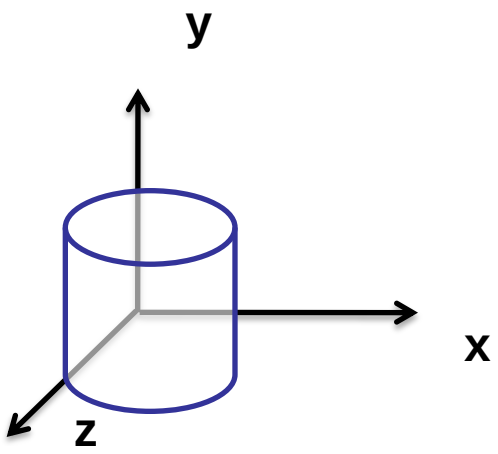
f : type of arguments (here float)

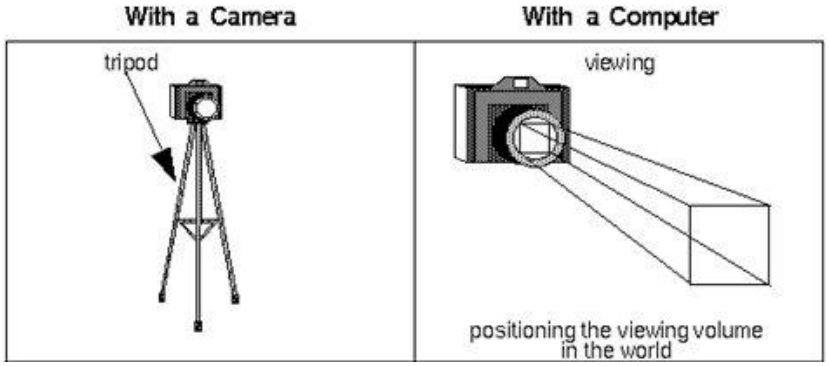
glColor4f, glColor3i, glVertex3f,...

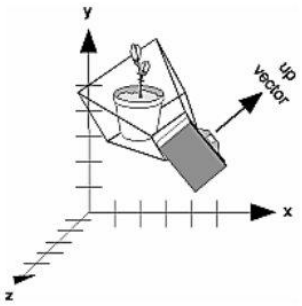
OpenGL Rendering Pipeline



Object Coordinate System and Camera







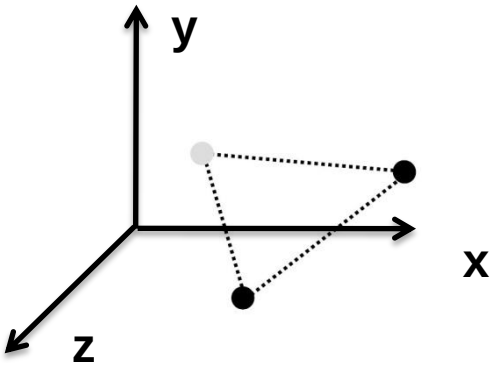
Object Coordinate Sytem

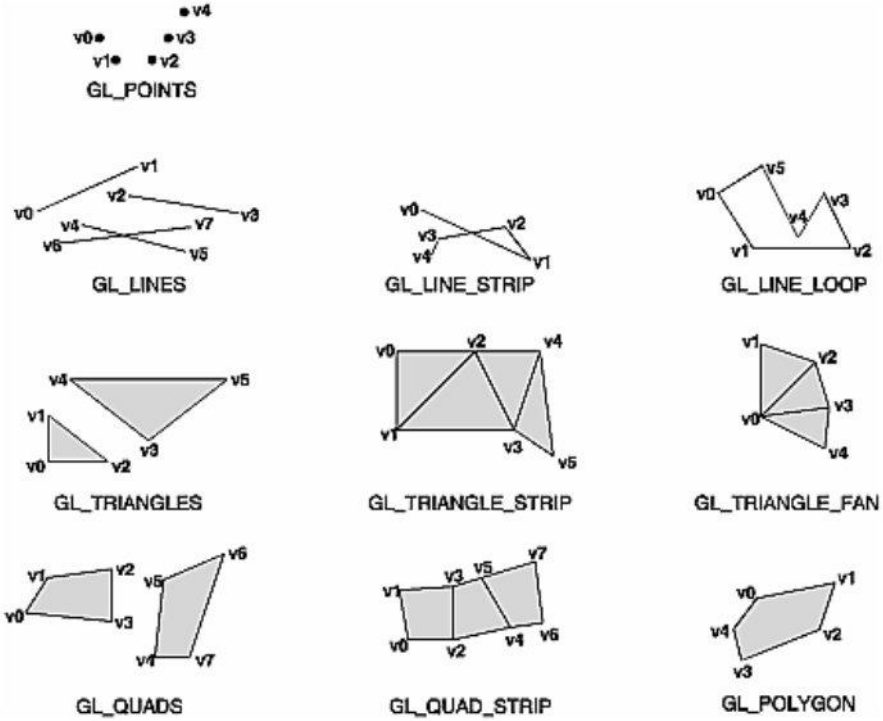
- Default Camera:
  - Camera –Position (eye) (0,0,0)
  - View-Vector (center) (0,0,-1)
  - Up-Vector (up) (0,1,0)
- Set Camera with `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`

Basic Primitives

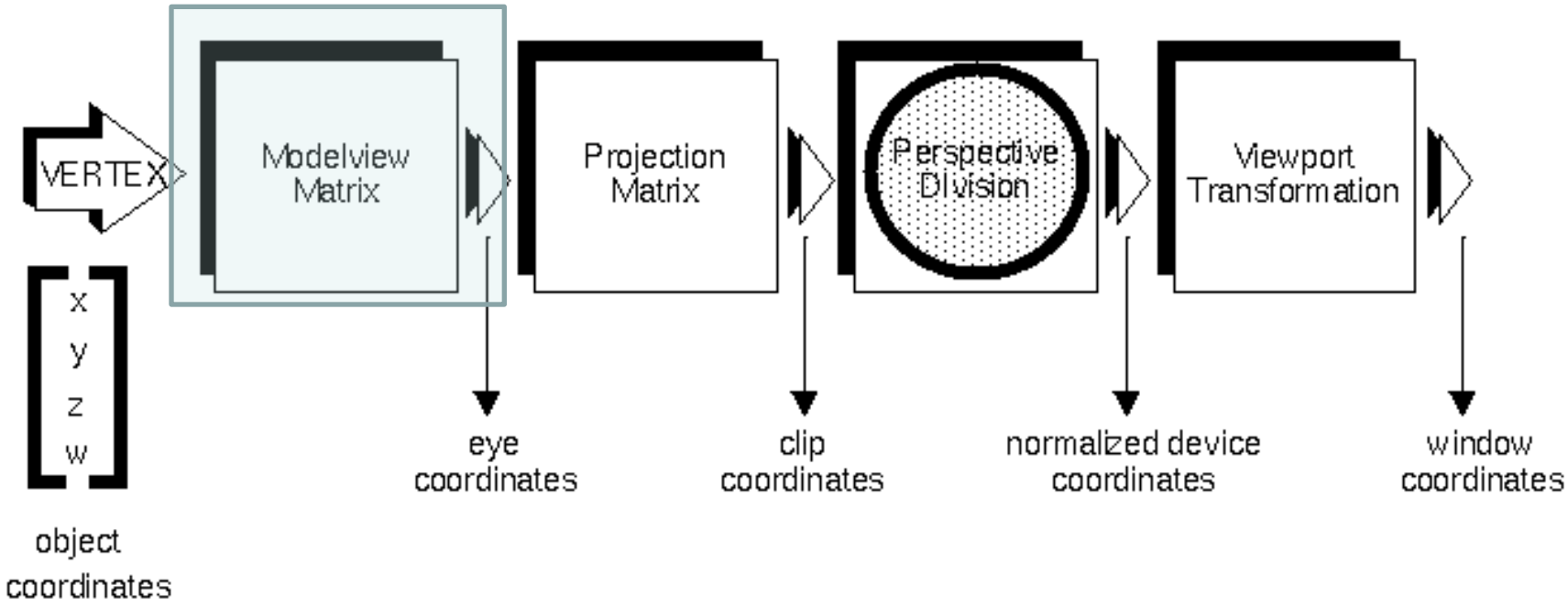
Draw Primitive:

```
glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, -1.0);
glVertex3f(0.0, 3.0, -1.0);
glVertex3f(4.0, 3.0, -1.0);
glVertex3f(6.0, 1.5, -1.0);
glVertex3f(4.0, 0.0, -1.0);
glEnd();
```





OpenGL Rendering Pipeline



3D Transformations in OpenGL

- Translation

```
glTranslatef(7.0f,0.0f,0.0f);          // Translate 7 units to the right
                                       // (positiv x-axis)
```

- Rotation

```
glRotatef(35.0f,0.0f,0.0f,1.0f);       // Rotate 35 Degrees around the z-
                                       // axis (counter-clockwise)
```

- Scale

```
glScalef(0.5f,0.5f,0.5f);              // Scale along x-,y- and z-axis by a
                                       // factor of 0.5
```

- Directly applying transformation matrix

```
glMultMatrixf(pointerToMatrix);        // Multiply current matrix with a 4x4
                                       // Matrix pointed by pointerToMatrix
```

- Commands generate a 4x4 matrix and multiply it with the current modelview matrix

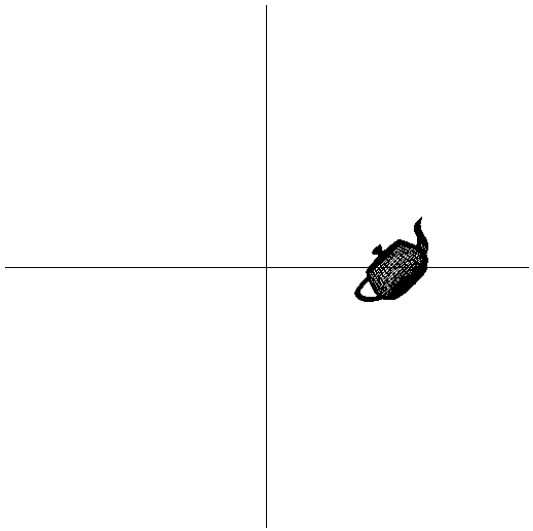
- Attention:
  - OpenGL stores matrices in column-major order

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

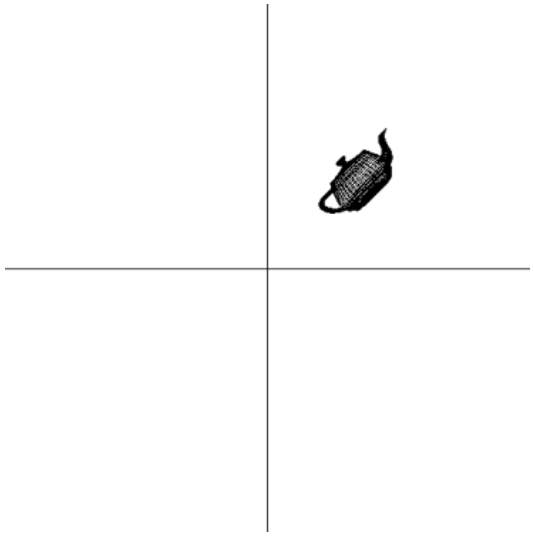
Order of transformations

- A(BC) = (AB)C, order of multiplication is irrelevant
- AB != BA, order of transformations is relevent

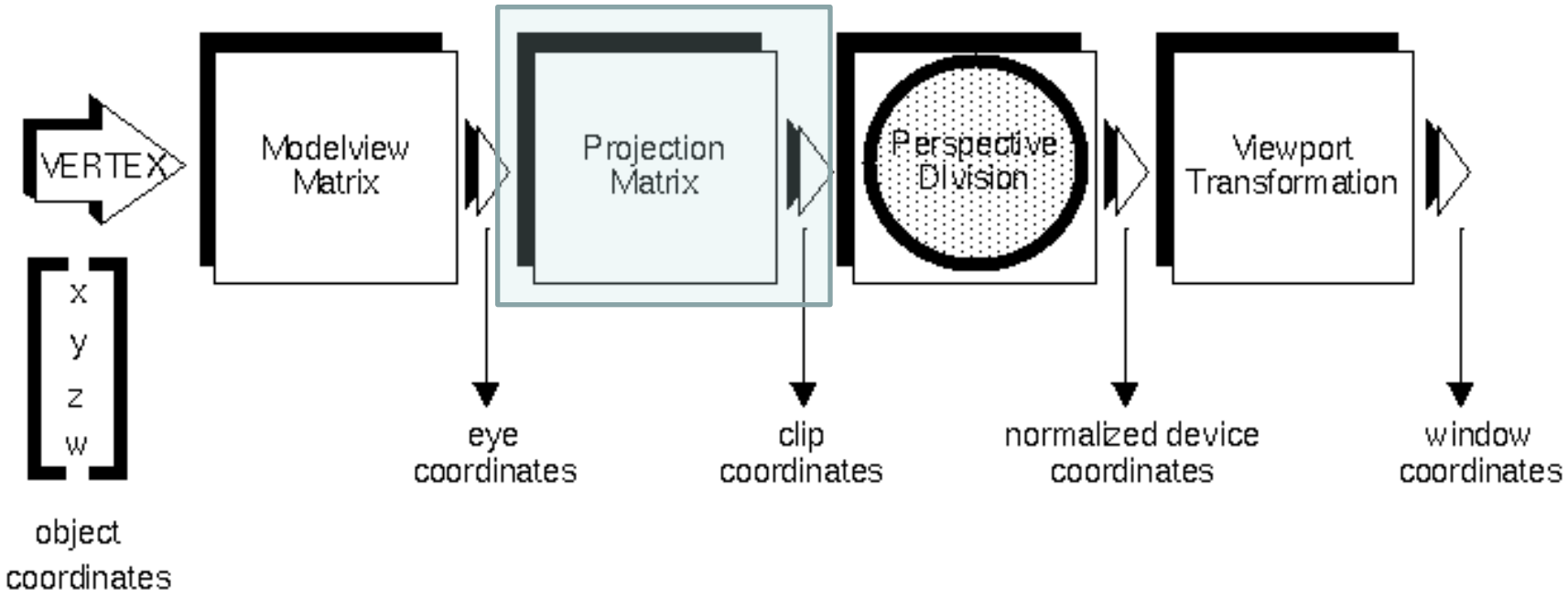
```
glTranslatef( 2.f, 0.f, 0.f );
glRotatef( 45.f, 0.f, 0.f, 1.f );
```



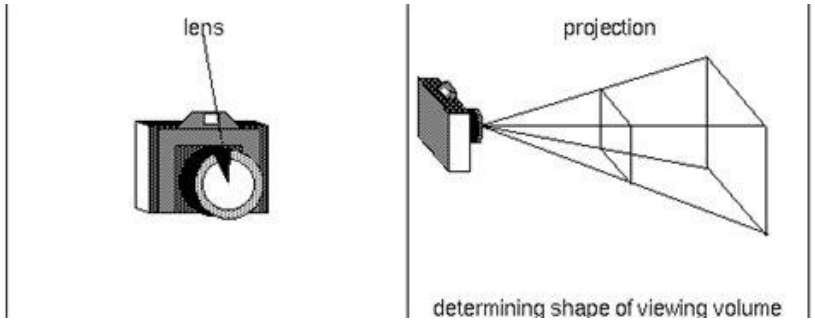
```
glRotatef( 45.f, 0.f, 0.f, 1.f );
glTranslatef( 2.f, 0.f, 0.f );
```

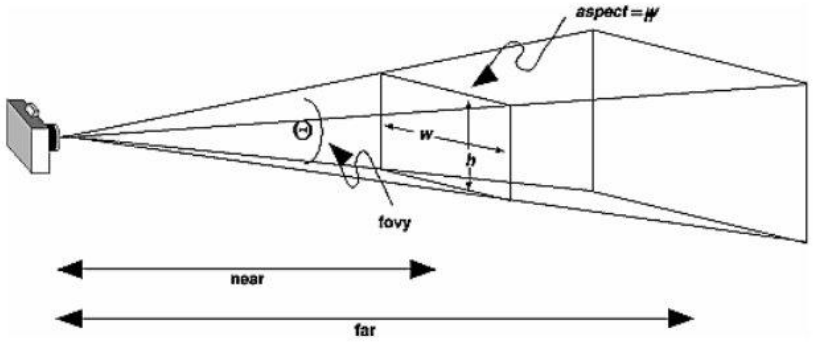


OpenGL Rendering Pipeline



Projection



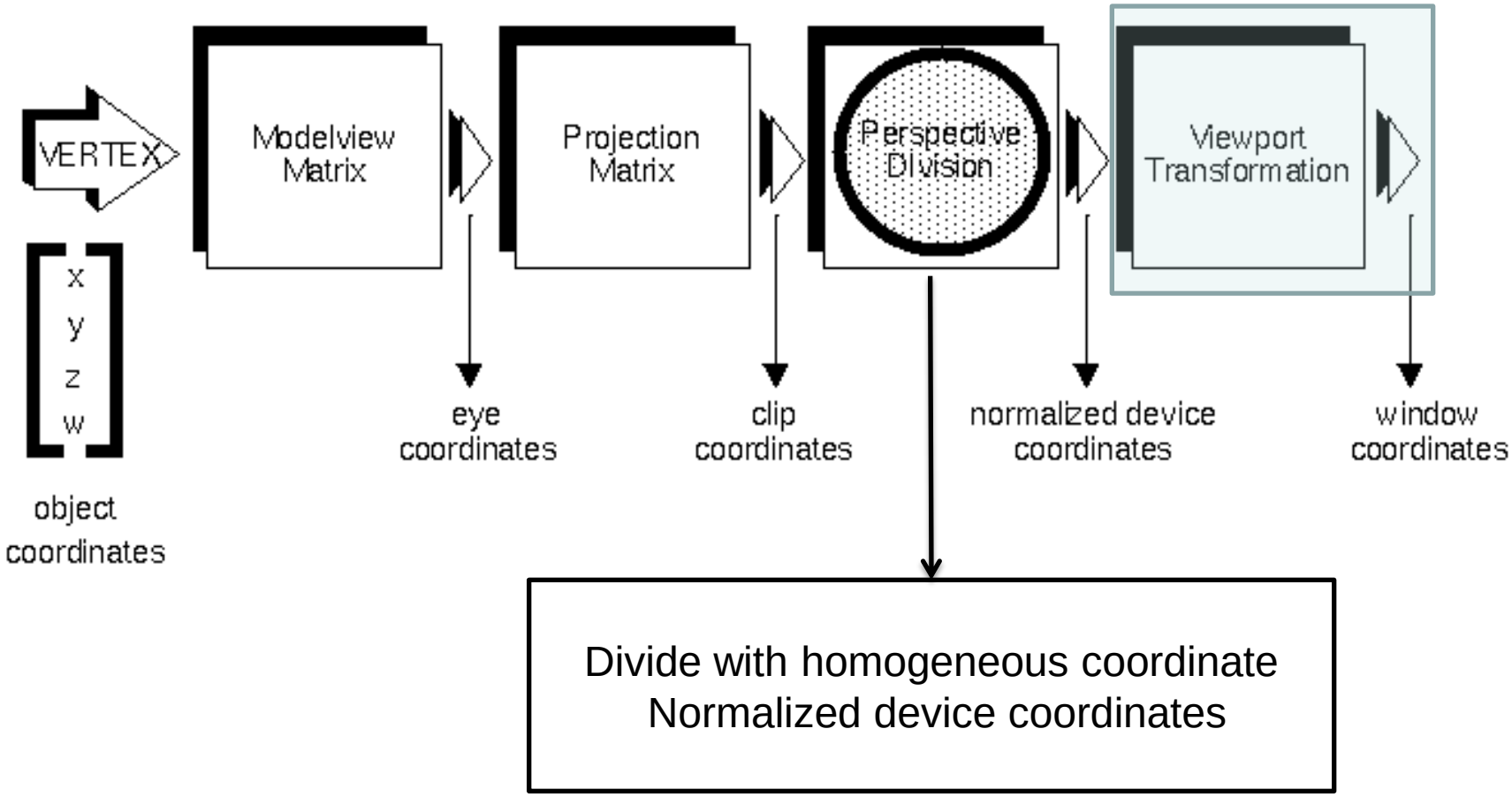


- Set Projection matrix

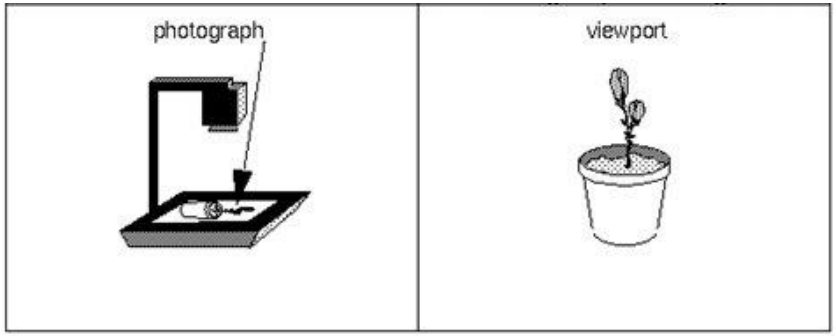
```
glMatrixMode(GL_PROJECTION); glLoadIdentity();
gluPerspective(fovy, aspect, near, far);
or
glFrustum(left, right, bottom, top, near, far);
or
glOrtho(left, right, bottom, top, near, far),
```

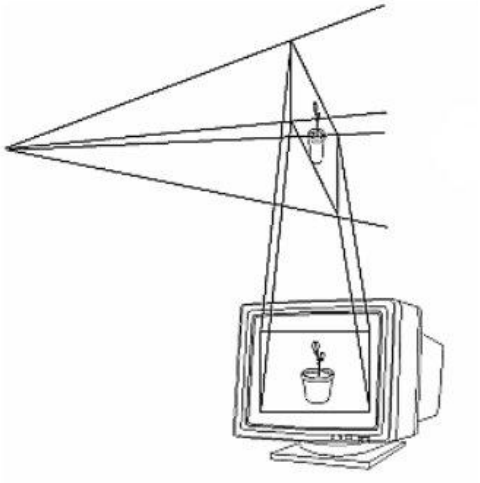
Attention: near >= 1.0, otherwise depth accuracy unacceptable

OpenGL Rendering Pipeline



Viewport





- Size of rendered image on monitor window (in pixels)

```
glViewport(x,y,width,height);
x,y: lower left corner of the window rectangle
width,height of the window rectangle
```

External helpers for models

- GLUT Objects
- `glutSolidSphere(GLdouble radius, GLint slices, GLint stacks), glutSolidCube(GLdouble size), glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks), glutSolidTeapot(GLdouble size)` and more
- Also available as wireframe (wireframe visualization can also be enabled through OpenGL calls)
  `glutWireTeapot(GLdouble size)`
- http://www.opengl.org/resources/libraries/glut/

OpenGL is a State-Machine

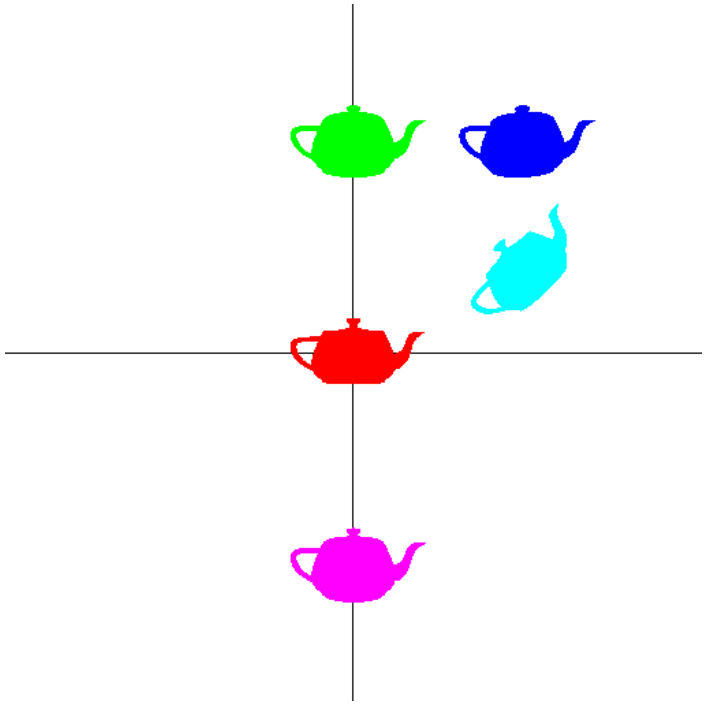
- Every change on the state affects subsequent calls and is only explicitly change with a new OpenGL call
  - E.g. . `glColor3f(1.0,0.0,0.0)`, the vertex color is set to red until stated otherwise
- The current state can be pushed onto a stack and recalled later
  - `glPushAttrib(GLbitfield mask)` pushes the state specified by `mark` on the stack
    e.g. `glPushAttrib(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` stores the current color and depth information on the stack
  - `glPopAttrib()` restores the state from the top of the stack
- Matrices can also be pushed on a separate stack
  - `glPushMatrix()` push current matrix
  - `glPopMatrix()` pop/restore matrix(basically set the current matrix to the one stored at the top of the matrix stack)

Direct operations on OpenGL matrices

- `glMatrixMode(matrix)`
  `matrix: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE, GL_COLOR`
  specify which matrix should be affected by subsequent operations
- `glLoadIdentity();`
  set current matrix to identity
- `glLoadMatrix{f,d}(pointerToMatrix);`
  set current matrix to the one stored in `pointerToMatrix`
- `glMultMatrix{f,d}(pointerToMatrix);`
  multiply current matrix with the one stored in `pointerToMatrix`
- `GLdouble model[16];`
  `glGetDoublev(GL_MODELVIEW_MATRIX,model);`
  read and store matrix (here modelview matrix) into `model`

Multiple Objects and Transformations

```
glPushMatrix();
        glColor3f(1.f,0.f,0.f);
        glutSolidTeapot(5.0); // Red teapot
        glTranslatef(0.0f,25.0f,0.0f);
        glColor3f(0.f,1.f,0.f);
        glutSolidTeapot(5.0); // Green teapot
        glTranslatef(20.0f,0.0f,0.0f);
        glColor3f(0.f,0.f,1.f);
        glutSolidTeapot(5.0); // Blue teapot
        glTranslatef(0.0f,-15.0f,0.0f);
        glRotatef(45.0f,0.0f,0.0f,1.0f);
        glColor3f(0.f,1.f,1.f);
        glutSolidTeapot(5.0); // Cyan teapot
glPopMatrix();
glPushMatrix();
        glColor3f(1.f,0.f,1.f);
        glTranslatef(0.0f,-25.0f,0.0f);
        glutSolidTeapot(5.0); // Magenta teapot
glPopMatrix();
```



Depthbuffer

- What happens if objects occlude each other?
- In the final image the objects would overlap
- Solution: Depthbuffer
- For each pixel a depth value is stored
- Depthfunc specifies condition for pixels passing the depth test

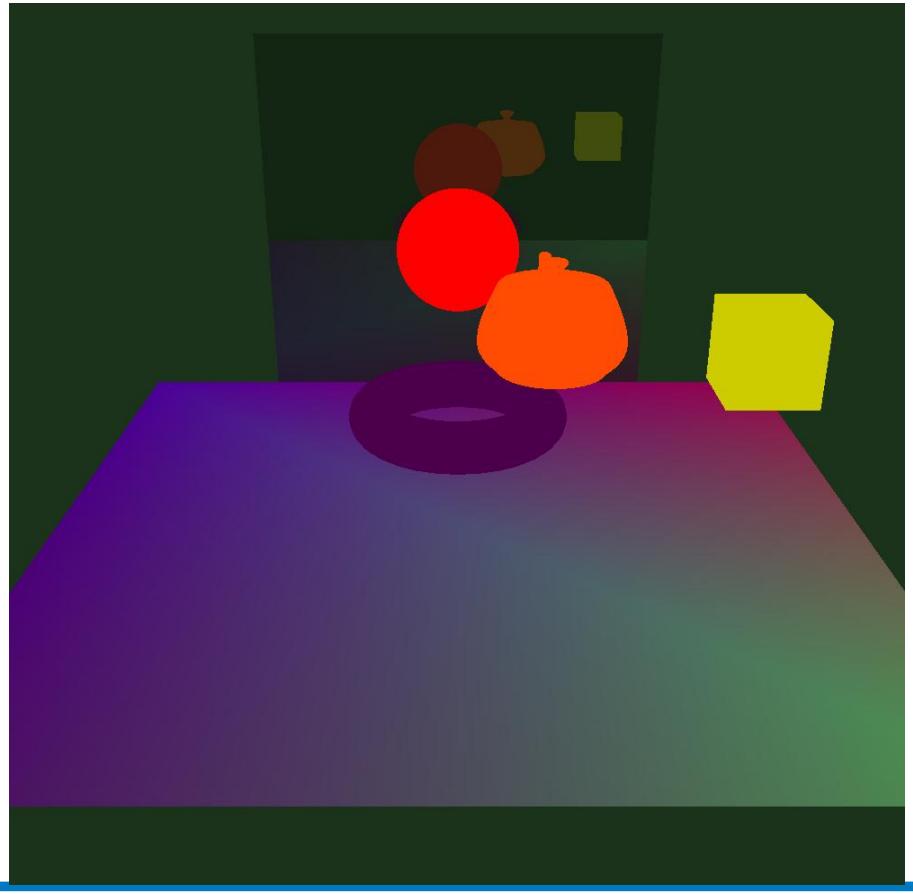
```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL); //default
```

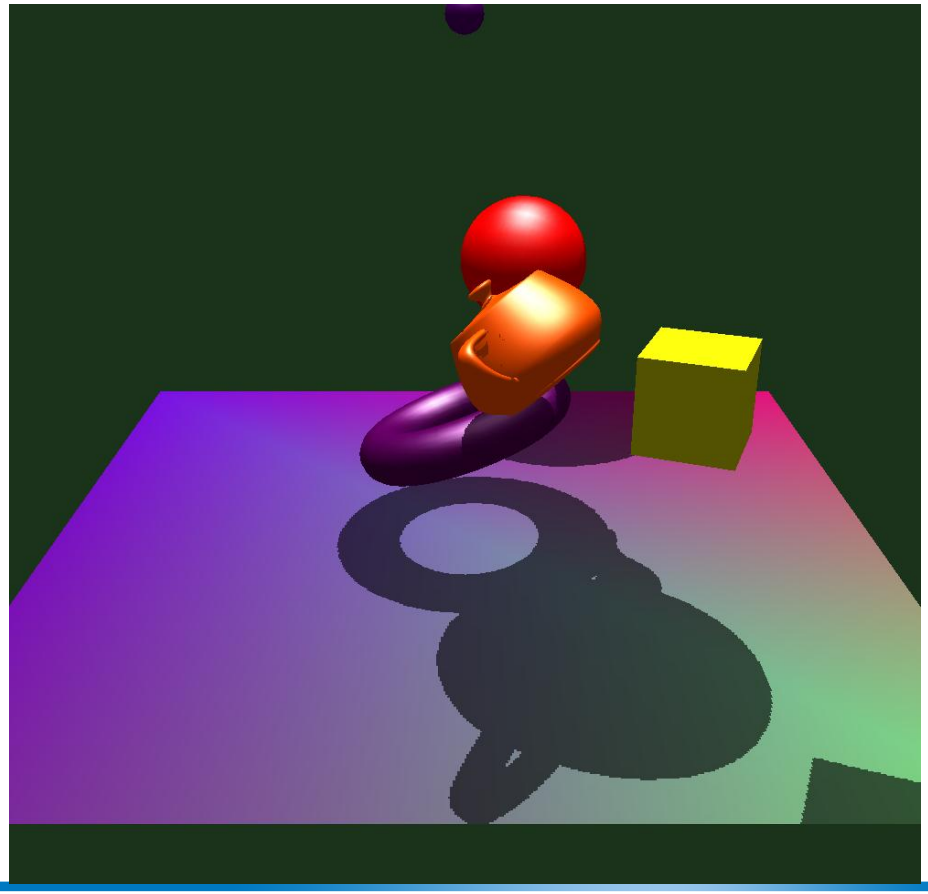
- GL_LEQUAL: The incoming pixel (fragment) passes the depth test if its depth is less or equal to the one stored in the buffer

Depth Test Disabled

Depth Test Enabled

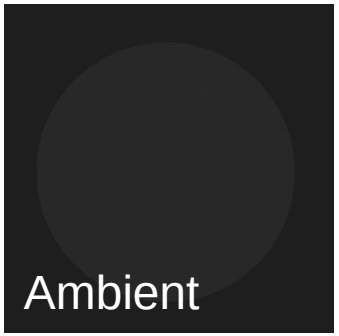
OpenGL Lighting

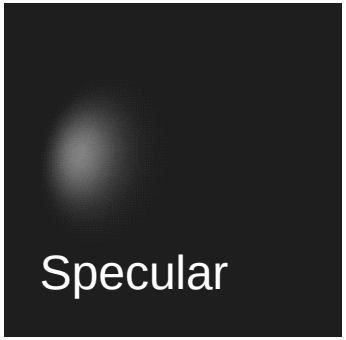




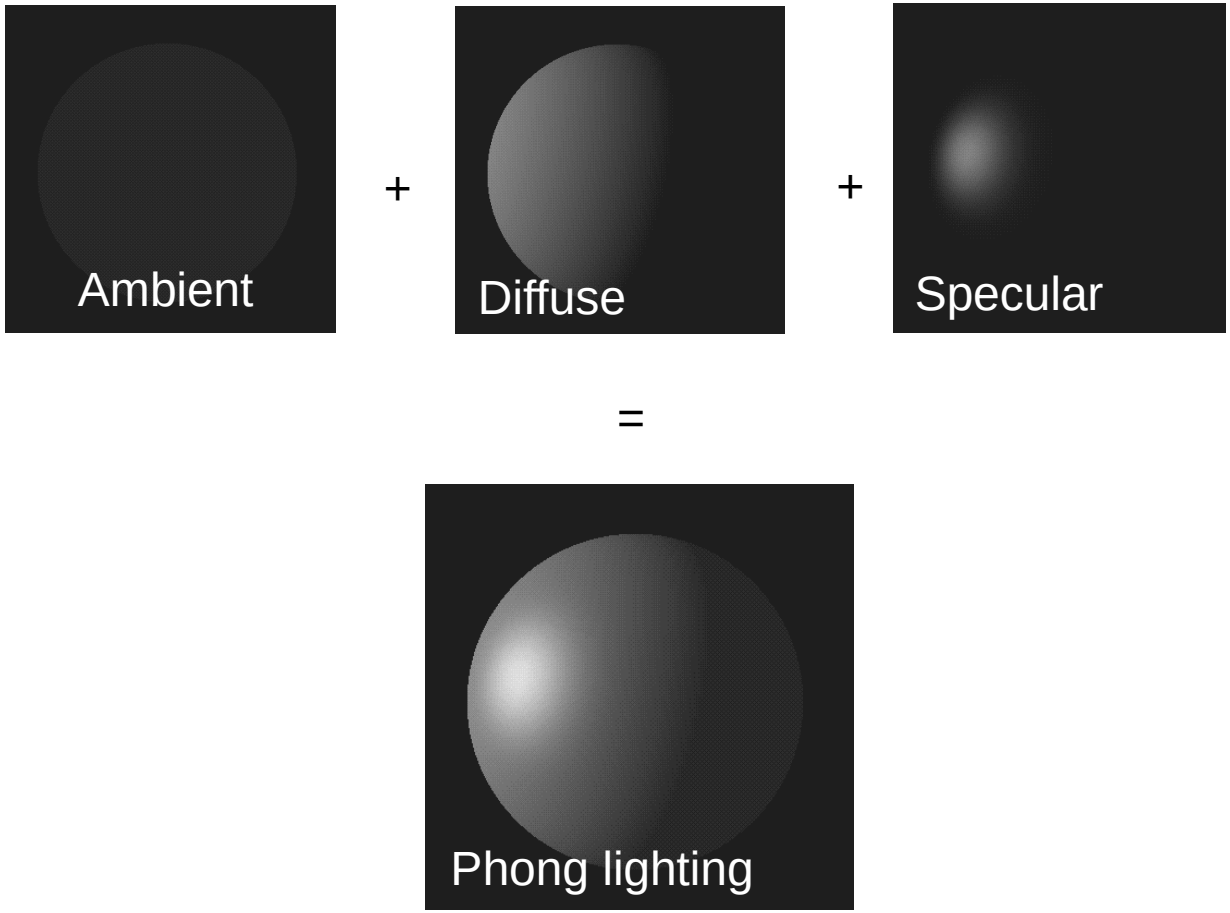
Different Types of Light

- Ambient Light
  - “Global” light emitted from a scene
-
- „ Diffuse Light”
  - Depends on surface normal + light direction
- „Specular Light“
  - Depends on surface normal + light direction + camera (eye) position
- „Emitted Light“
  - Light is „emitted“ from polygon





Phong Lighting Model



Light in OpenGL

```
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);
```

Maximal number of lights: `int maxnumlights = 0;`

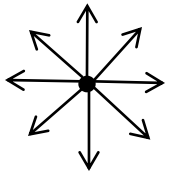
```
glGetIntegerv(GL_MAX_LIGHTS, &maxnumlights);
```

The performance decreases with increasing number of lights

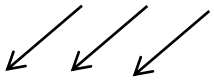
Light in OpenGL

```
GLfloat light_position[] = { 1.0, 1.0, 1.0, w };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Positional light source w>0



Directional light source w = 0



OpenGL is not a GUI Toolkit

- OpenGL does not offer functionality to generate and manage GUI windows
- OpenGL assumes that the programmer provides a valid GUI window together with an OpenGL render context
- Window and render context can be provided through different toolkits, such as: GLUT, Qt, FLTK, MFC,...

- GLUT – OpenGL Utility Library: A simple platform independent library for generating a window with OpenGL context.
  - Can also handle keyboard and mouse events.

- Websites http://www.opengl.org/resources/libraries/glut/ , http://openglut.sourceforge.net/ , http://www.xmission.com/~nate/glut.html

Setup an OpenGL App with GLUT

- Download GLUT source-code or the precompiled library for your platform (easier for beginners)
- Include header files glut.h und glu.h
- Include GLUT and GLU library as linker inputs (glut32.lib, glu32.lib)
- Most common error: Forgot to include header and libraries

Download http://www.opengl.org/resources/libraries/glut/ , http://openglut.sourceforge.net/ , http://www.xmission.com/~nate/glut.html

GLUT App – Min. requirements

```
1
2
3   #include <windows.h>    // Standard Header For Most Programs
4   #include <gl/gl.h>      // The GL Header File
5   #include <gl/glut.h>    // The GL Utility Toolkit (Glut) Header
6
7   void init ( GLvoid )   ;  // Create Some Everyday Functions
8   void display ( void ) ;  // Create The Display Function
9   void reshape ( int w, int h ) ;  // Create The Reshape Function (the viewport)
10  void keyboard ( unsigned char key, int x, int y );  // Create Keyboard Function
11
12  void main ( int argc, char** argv )    // Create Main Function For Bringing It All Together
13  {
14    glutInit            ( &argc, argv ); // Erm Just Write It =)
15    init ();
16    glutInitDisplayMode ( GLUT_RGB | GLUT_DOUBLE ); // Display Mode
17    glutInitWindowSize  ( 500, 500 ); // If glutFullScreen wasn't called this is the window size
18    glutCreateWindow    ( "FOOBAR" ); // Window Title (argv[0] for current directory as title)
19    glutFullScreen      ( );          // Put Into Full Screen
20    glutDisplayFunc     ( display );  // Matching Earlier Functions To Their Counterparts
21    glutReshapeFunc     ( reshape );
22    glutKeyboardFunc    ( keyboard );
23    glutMainLoop        ( );          // Initialize The Main Loop
24
25  }
26
```

GLUT App – Min. requirements

```
27
28  void init ( GLvoid )      // Create Some Everyday Functions
29  {
30      glShadeModel(GL_SMOOTH);                          // Enable Smooth Shading
31      glClearColor(0.0f, 0.0f, 0.0f, 0.5f);             // Black Background
32      glClearDepth(1.0f);                               // Depth Buffer Setup
33      glEnable(GL_DEPTH_TEST);                          // Enables Depth Testing
34      glDepthFunc(GL_LEQUAL);                           // The Type Of Depth Testing To Do
35      glEnable ( GL_COLOR_MATERIAL );
36      glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
37
38      ........................
39  }
40
```

GLUT App – Min. requirements

```
41  void display ( void )    // Create The Display Function
42  {
43    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);    // Clear Screen And Depth Buffer
44      glLoadIdentity();                                    // Reset The Current Modelview Matrix
45      glTranslatef(-1.5f,0.0f,-6.0f);                      // Move Left 1.5 Units And Into The Scre
46      glBegin(GL_TRIANGLES);                               // Drawing Using Triangles
47          glVertex3f( 0.0f, 1.0f, 0.0f);                   // Top
48          glVertex3f(-1.0f,-1.0f, 0.0f);                   // Bottom Left
49          glVertex3f( 1.0f,-1.0f, 0.0f);                   // Bottom Right
50      glEnd();                                             // Finished Drawing The Triangle
51      glTranslatef(3.0f,0.0f,0.0f);                        // Move Right 3 Units
52      glBegin(GL_QUADS);                                   // Draw A Quad
53          glVertex3f(-1.0f, 1.0f, 0.0f);                   // Top Left
54          glVertex3f( 1.0f, 1.0f, 0.0f);                   // Top Right
55          glVertex3f( 1.0f,-1.0f, 0.0f);                   // Bottom Right
56          glVertex3f(-1.0f,-1.0f, 0.0f);                   // Bottom Left
57      glEnd();
58
59
60    glutSwapBuffers ( );
61    // Swap The Buffers To Not Be Left With A Clear Screen
62
63    ......................
64  }
```

GLUT App – Min. requirements

```
66  void reshape ( int w, int h )    // Create The Reshape Function (the viewport)
67  {
68    glViewport     ( 0, 0, w, h );
69    glMatrixMode   ( GL_PROJECTION );  // Select The Projection Matrix
70    glLoadIdentity ( );                // Reset The Projection Matrix
71    if ( h==0 )  // Calculate The Aspect Ratio Of The Window
72       gluPerspective ( 80, ( float ) w, 1.0, 5000.0 );
73    else
74       gluPerspective ( 80, ( float ) w / ( float ) h, 1.0, 5000.0 );
75    glMatrixMode   ( GL_MODELVIEW );  // Select The Model View Matrix
76    glLoadIdentity ( );     // Reset The Model View Matrix
77
78      ..........................
79  }
```

GLUT App – Min. requirements

```
80
81  void keyboard ( unsigned char key, int x, int y )   // Create Keyboard Function
82  {
83    switch ( key ) {
84      case 27:          // When Escape Is Pressed...
85        exit ( 0 );     // Exit The Program
86        break;          // Ready For Next Case
87      default:          // Now Wrap It Up
88        break;
89    }
90
91      ........................
92  }
```

This was an introduction, OpenGL has a LOT more to offer

- For more info see:
- The Red Book
  Version 1.1 is online http://www.opengl.org/documentation/red_book/
- NeHe Online Tutorials including code http://nehe.gamedev.net/
- The OpenGL specification (very technical, but includes everything)
  http://www.opengl.org/
- GLUT Specification
  http://www.opengl.org/resources/libraries/glut/spec3/spec3.html
- OpenGL Interactive Tutorials
  http://www.xmission.com/~nate/tutors.html
- Common OpenGL Pitfalls
  http://www.opengl.org/resources/features/KilgardTechniques/oglpitfall/
- www.GameDev.net, Lots of useful articles and active developer community