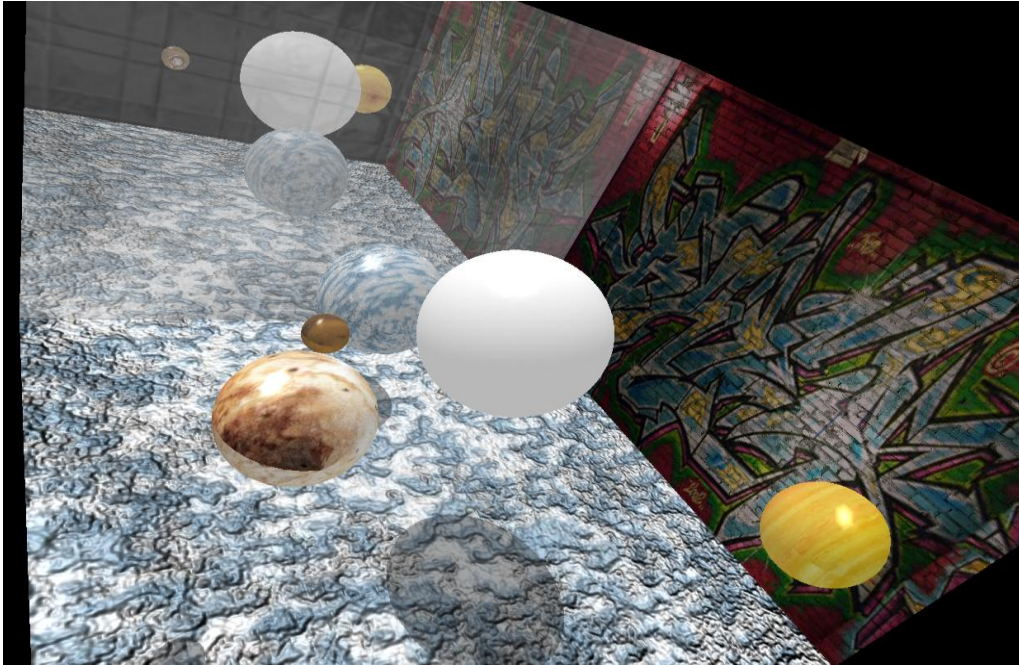
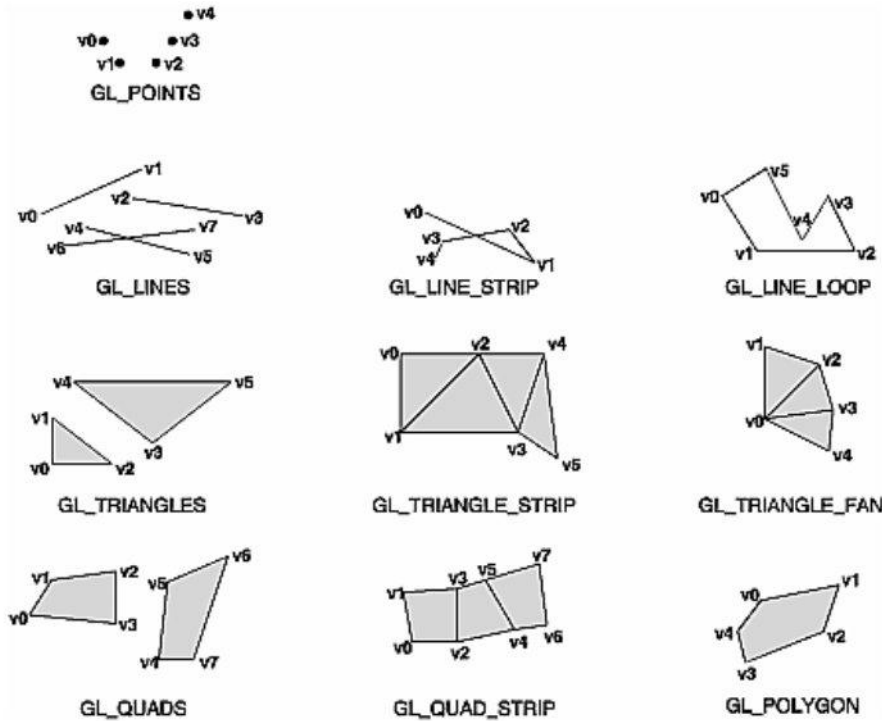


Erste Schritte in OpenGL



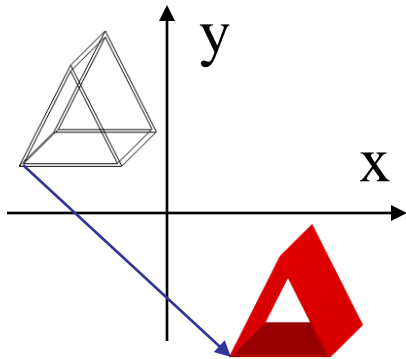
Modellierung – Aus Punkten werden geometrische Körper



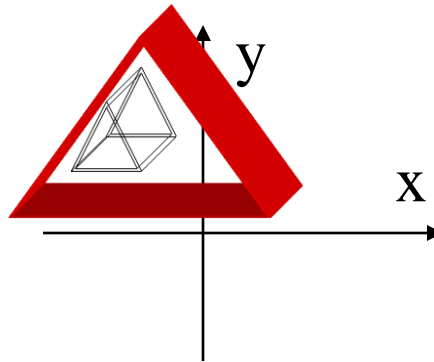
- Komplexe Objekte werden aus mehreren einfachen Objekten erstellt
- Dazu müssen Vertexes (Punkte) im virtuellen Raum angeordnet werden

3D Transformation

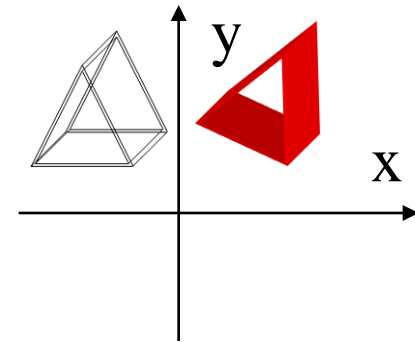
Drei der wichtigsten Koordinatentransformationen



Translation



Skalierung



Rotation

3D Translation & Rotation in kartesischen Koordinaten

3D Punkt

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

Translation

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

Rotation um die Z-Achse

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

Aufeinanderfolgende Transformationen erfordern die Kombination von
Matrixmultiplikation & Vektoraddition

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

Homogenen Koordinaten

- Homogene Koordinaten ermöglichen es auch die Translation als Matrixmultiplikation anzuwenden.
- Der Raum wird dazu um eine Dimension erweitert.
- Translation kann ebenfalls als Matrix dargestellt werden.

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

Translationsmatrix

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

Translation

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

Homogene Koordinaten

Rotationsmatrix

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

Rotation

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

Aus $\vec{p}' = M_{3 \times 3} \vec{p} + t$ wird $\vec{p}' = M_{4 \times 4} \vec{p}$

Somit lassen sich alle Transformationen im Raum durch Matrizen beschreiben.

Beispiel 3D Euklidische Transformation

Zuerst Rotation um die Z-Achse dann Translation
(Matrixmultiplikation von rechts nach links)

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

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

3D Skalierung

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

Euler-Winkel für die Rotation $R=R_zR_yR_x$

Rotation ψ um die z-Achse:

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

Rotation θ um die y-Achse:

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

Rotation ϕ um die x-Achse:

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

OpenGL (Graphics Library) seit 1992

- API Spezifikation für Plattform-, Programmiersprachen-, Hardware-unabhängigen Zugriff auf (beschleunigte) Graphikhardware
 - Entwickelt von Silicon Graphics
 - Dessen Implementierung wird i.A. mit dem Graphikkartentreiber mitgeliefert.
 - Zustandsbasiertes Modell
 - Wozu: Visualisierung, Computergraphik, Spielentwicklung, ...
 - Aktuelle Version 3.0 (August 2008)
 - Erweiterbar mit Hardwarehersteller spezifischen Extensions die über Extensionfunktionen nutzbar sind (siehe GLEW OpenGL Extension Wrangler Library)
-
- Websites <http://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

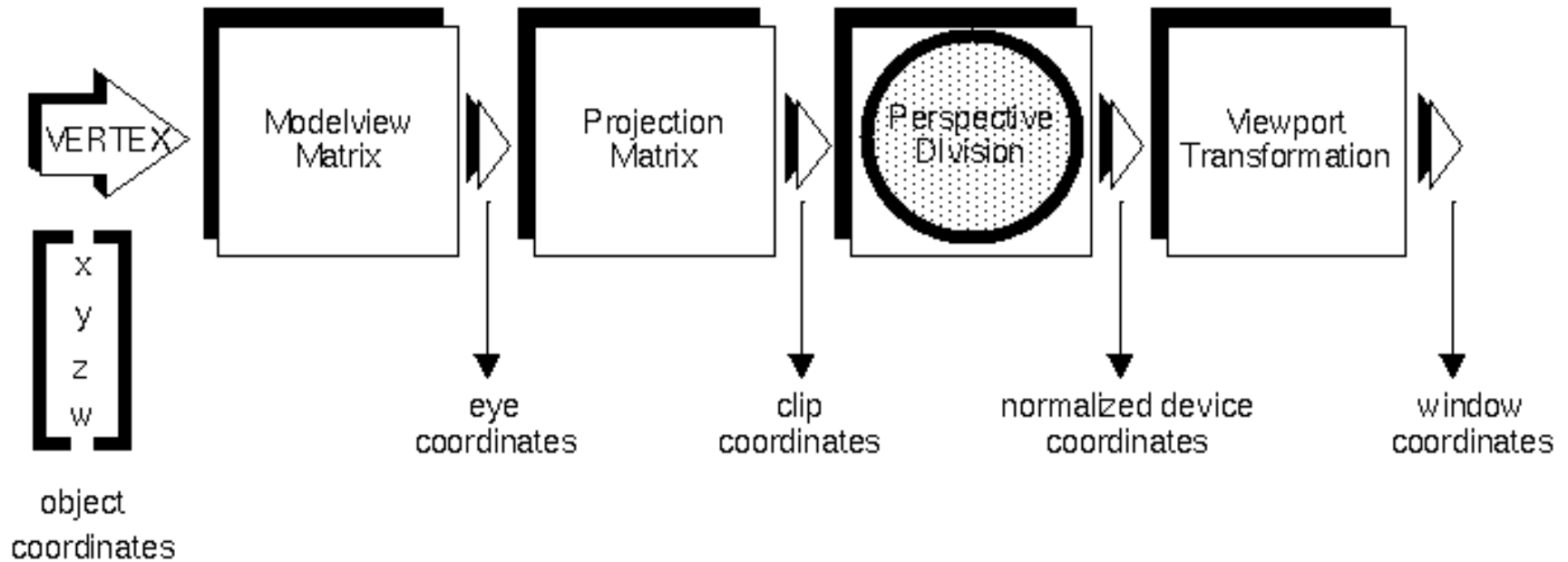
Color: Main Command

3 : number of arguments

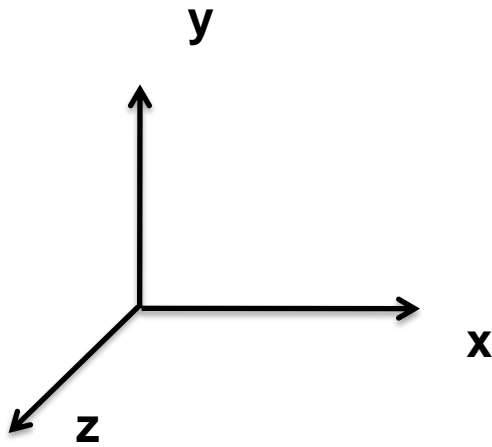
f : type of arguments (here float)

glColor4f, glColor3i, glVertex3f,...

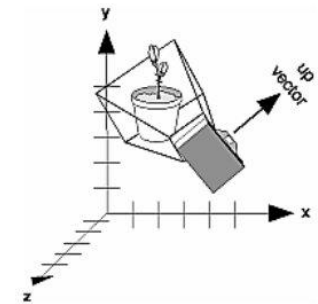
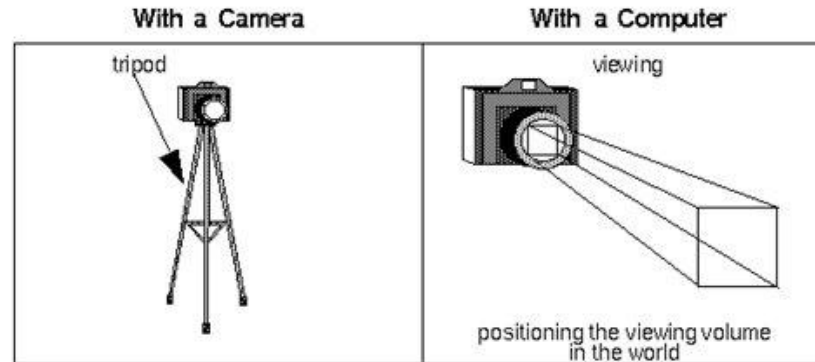
Rendering Pipeline für 3D



Objekt/Welt Koordinaten Sytem und Kamera

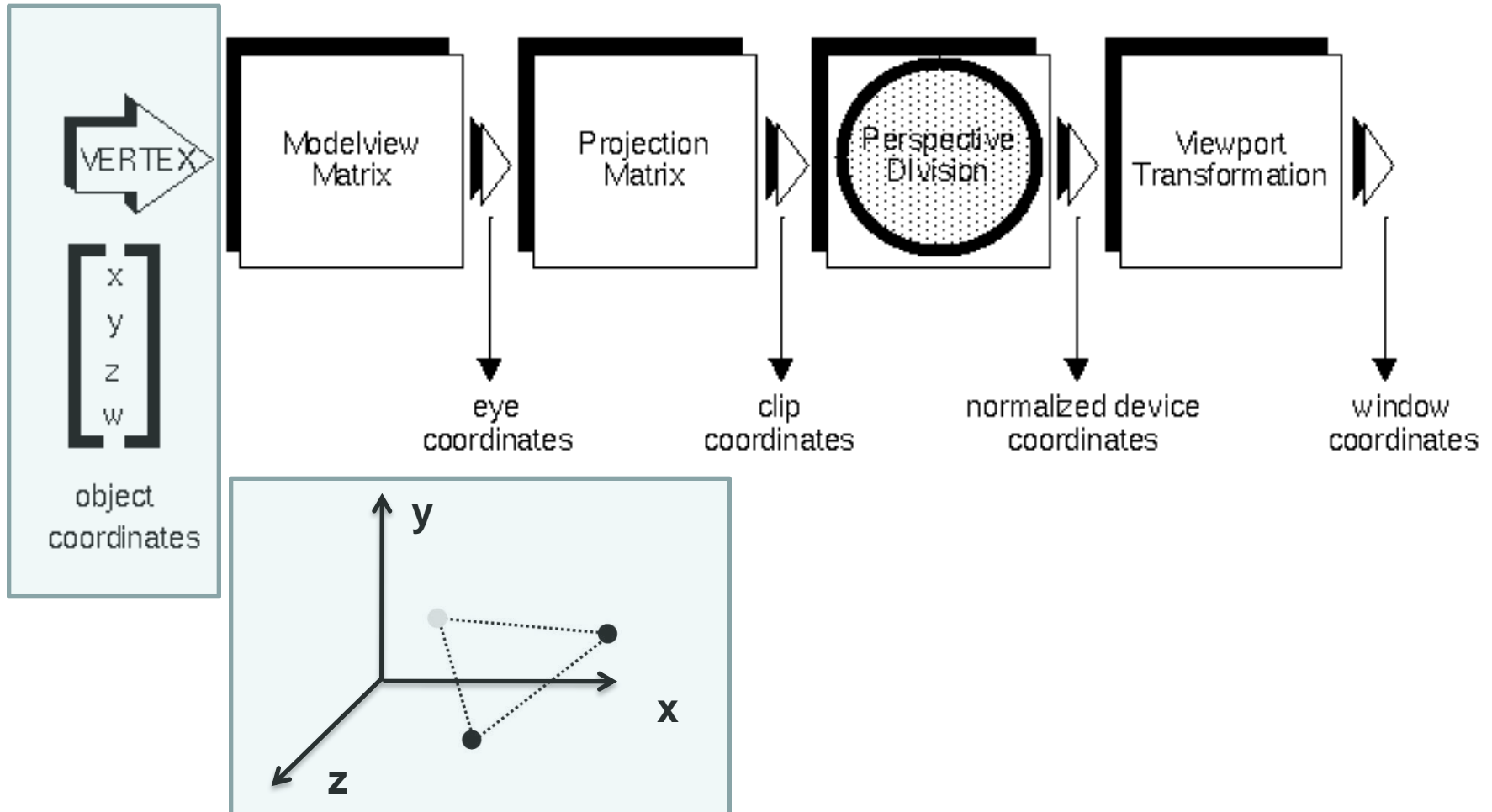


World Coordinate Sytem

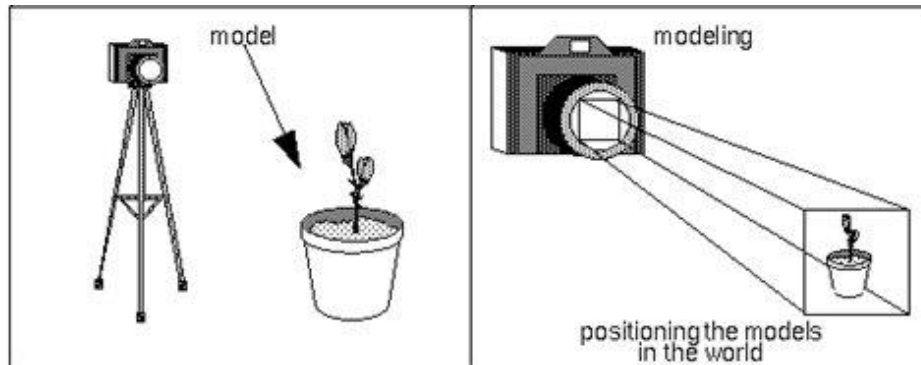


- Kameraposition mit `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`
- Default-Kamera: Position auf $(0,0,0)$ Sicht-Vector $(0,0,-1)$, Up-Vector $(0,1,0)$

Rendering Pipeline für 3D

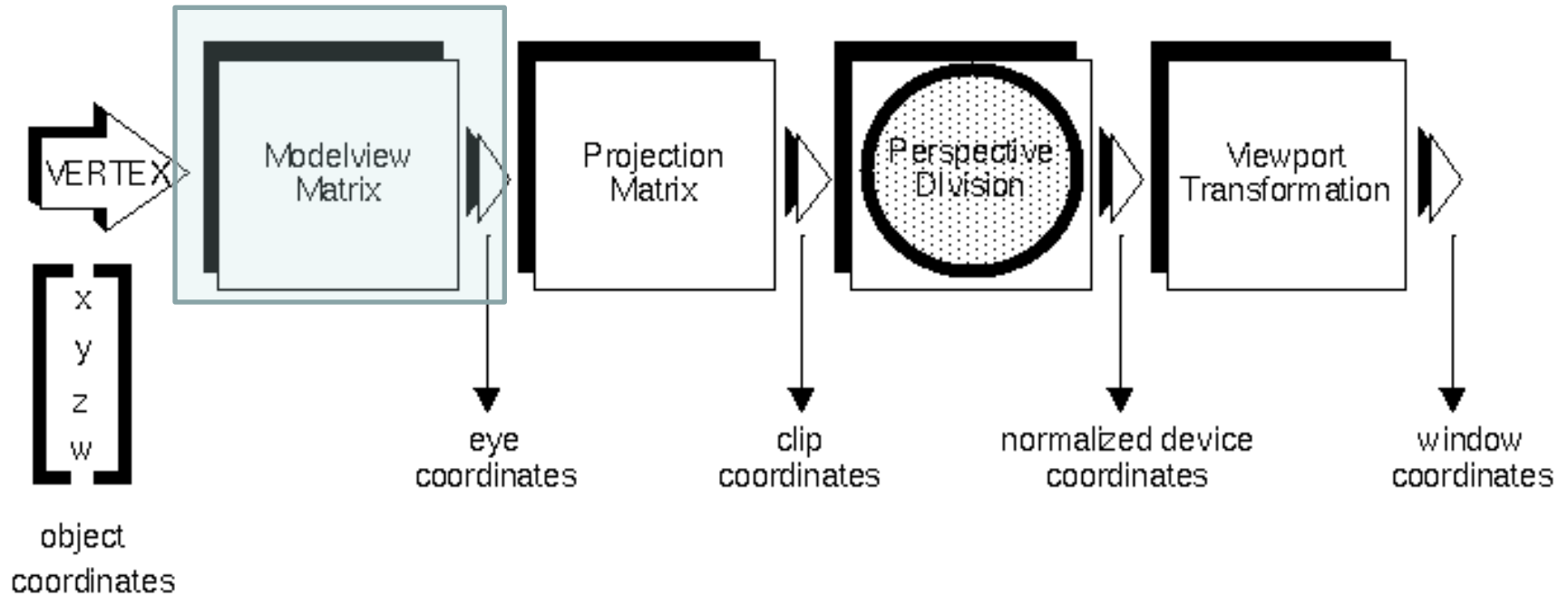


Modelieren



- **Modell generieren** `glBegin()... glEnd()`
oder `glut[Object]`, `glu[Object]`, eigene
Modelle/Renderobjects
- **Modell befindet sich im Welt-Koordinaten Raum**

Rendering Pipeline für 3D



3D Transformationen in OpenGL

- Translation

```
glTranslatef(7.0f,0.0f,0.0f);
```

```
// 7 Einheiten nach rechts verschieben  
// (in positiver x-Richtung)
```

- Rotation

```
glRotatef(35.0f,0.0f,0.0f,1.0f);
```

```
// 35 Grad um die z-Achse rotieren  
// (Gegen den Uhrzeigersinn)
```

- Skalierung

```
glScalef(0.5f,0.5f,0.5f);
```

```
// in x-, y- und z-Richtung  
// um Faktor 0.5 skalieren
```

- Transformation in homogenen Koordinaten

```
glMultMatrixf(pointerToMatrix);
```

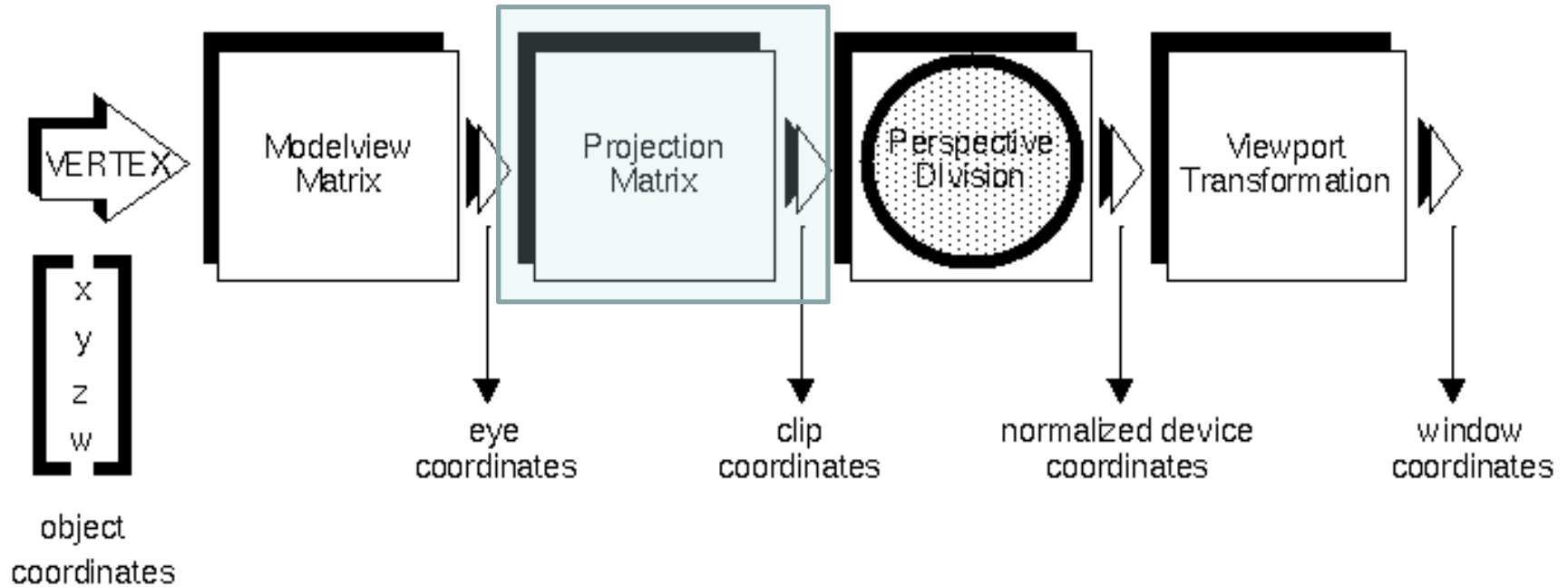
```
// Multipliziert die aktuelle 4x4  
// Matrix mit der 4x4 Matrix, auf die  
// von pointerToMatrix gezeigt wird
```

- Nicht vergessen:

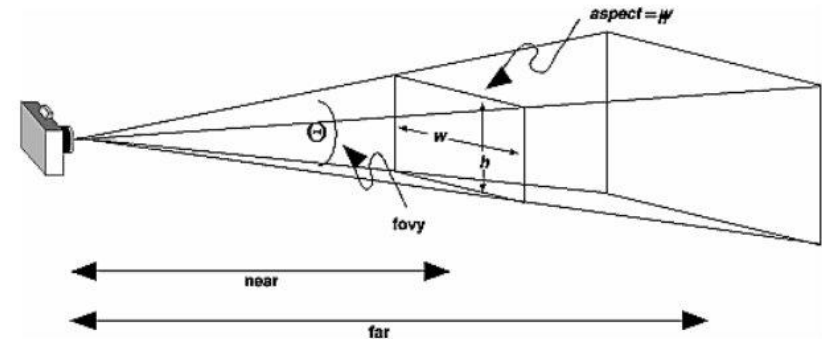
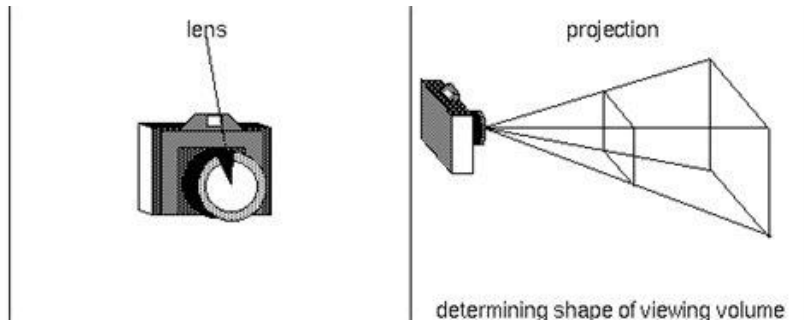
- In OpenGL werden die neuen Matrizen von rechts multipliziert
- In OpenGL sind Matrizen im Speicher spaltenweise abgelegt:

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

Rendering Pipeline für 3D



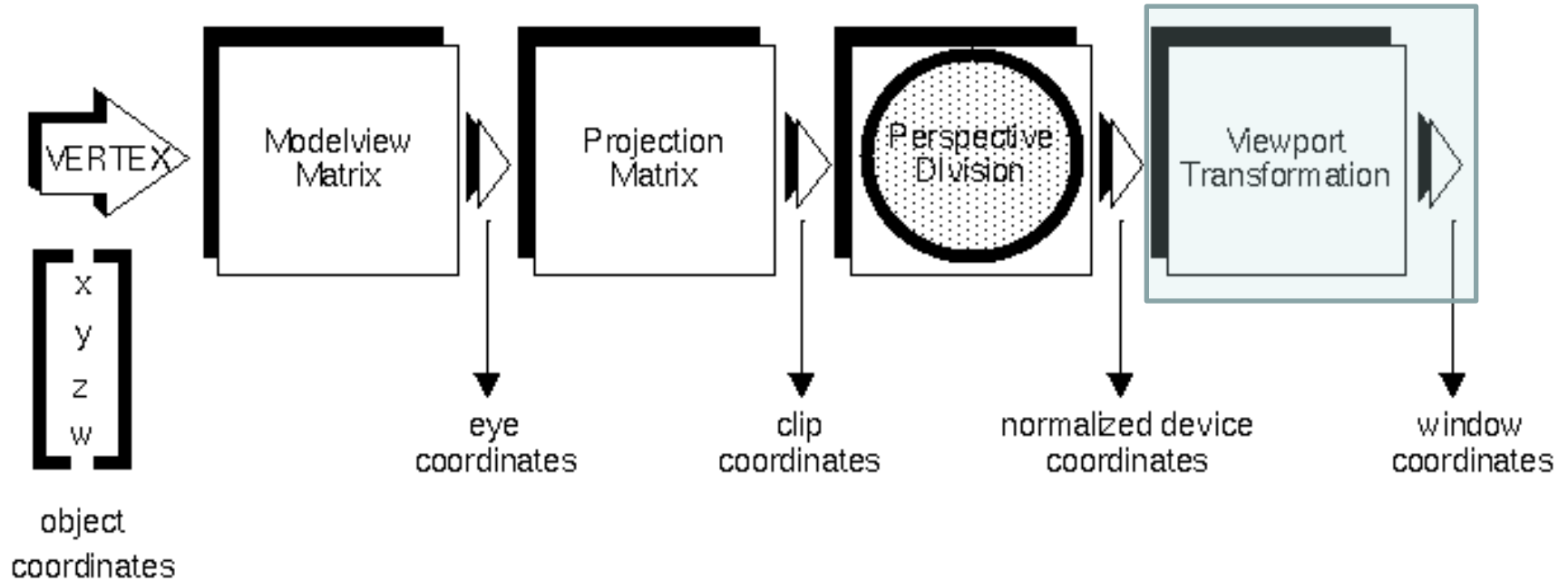
Projektion



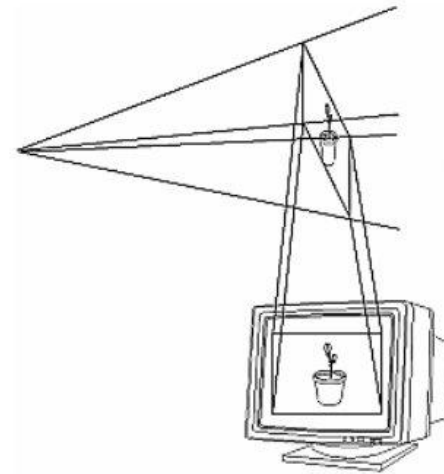
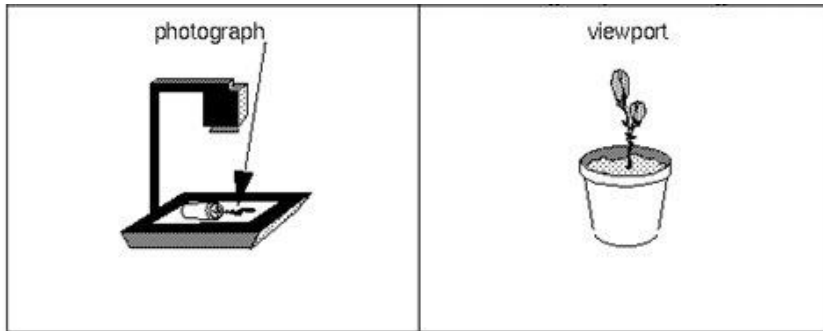
- Projektionsparameter setzen

```
glMatrixMode(GL_PROJECTION); glLoadIdentity();  
gluPerspective(45.0, 1.0, 1.0, 1000.0);  
oder glFrustum(...); glOrtho(),
```

Rendering Pipeline für 3D



Viewport

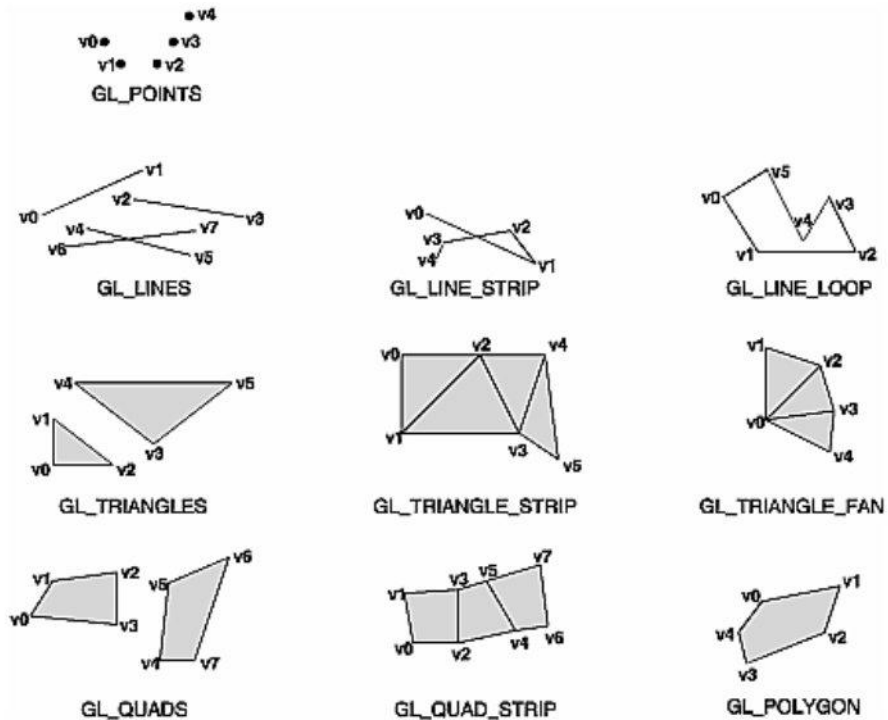


- Bildauschnitt setzen
`glViewport(0, 0, width, height);`

Grundobjekte

Primitive zeichnen:

```
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();
```



- GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, GL_POLYGON

Grundobjekte

- Außer mit `glBegin(...)` und `glEnd()` können mit GLUT einfach einige Grundobjekte gezeichnet werden
<http://www.opengl.org/resources/libraries/glut/>
- `glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)`, `glutSolidCube(GLdouble size)`,
`glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks)`, `glutSolidTeapot(GLdouble size)`
- Alle GLUT Objekte können auch als Drahtmodell gezeichnet werden z.B. `glutWireTeapot(GLdouble size)`

OpenGL ist ein Zustandsautomat

- D.h. wenn man Sachen ändert, bleiben sie solange bestehen, bis man sie wieder ändert.
- Man kann einen Zustand in den Keller [Stack] „retten“, um ihn später wiederherzustellen
 - `glPushAttrib(GL_EIGENSCHAFT)` rettet den Wert von `GL_EIGENSCHAFT` in den Keller [Stack]
 - `glPopAttrib(GL_EIGENSCHAFT)` stellt den Wert wieder her
 - `glPushAttrib(GL_ALL_ATTRIB_BITS)` rettet alle Attribute / Eigenschaften
 - z.B. `glColor3f(1.0,0.0,0.0)`, Color is set to red and stays until new call to `glColor()`
- Das gleiche gilt auch für die Matrizen
 - `glPushMatrix()` rettet die aktuelle Matrix
 - `glPopMatrix()` stellt die gerettete Matrix wieder her

Matrix Mode :: MODELVIEW_MATRIX

- `glLoadIdentity();`
setzt die Modelview-Matrix auf die Identität
- `glLoadMatrix*(pointerToMatrix);`
setzt den Wert der Modelview-Matrix auf die `pointerToMatrix`
- `glMultMatrix*(pointerToMatrix);`
multipliziert den Wert der Modelview-Matrix mit `pointerToMatrix`
- `GLdouble model[16];`
`glGetDoublev(GL_MODELVIEW_MATRIX, model);`
liest den Inhalt der Modelview-Matrix in die Variable `model`

* ist ein Platzhalter fuer den Datentyp. Exemplarische Werte sind (i) 32-bit integer, (f) 32-bit floating-point or (d) 64-bit floating-point double

Adventskranz neben Teekessel

```
void adventskranz() {
    glPushMatrix();
        glutSolidTorus(6, 10, 10, 10);
        glTranslatef(-5.0,0,5.0); // Verschiebung nach vorn rechts
        glutSolidCone(5.0, 8.0, 10, 10); //Breite 5, Höhe 8
        glTranslatef(10.0,0.0,0.0); // Verschiebung nach rechts
        glutSolidCone(5.0, 8.0, 10, 10); //Breite 5, Höhe 8
        glTranslatef(0.0,0.0,-10.0); // Verschiebung von Kamera weg
        glutSolidCone(5.0, 8.0, 10, 10); //Breite 5, Höhe 8
        glTranslatef(-10.0,0.0,0.0); // Verschiebung nach links
        glutSolidCone(5.0, 8.0, 10, 10); //Breite 5, Höhe 8
    glPopMatrix();
}
```

```
glPushMatrix();
    adventskranz();
glPopMatrix();
glTranslate(-20.0,0.0,0.0);
glutSolidTeapot(10.0);
```

Tiefenpuffer

- Neben dem Framebuffer (Bild Puffer) für Farbwerte für jedes Pixel auch Puffer für Tiefenwert. Wird dieser aktiviert, wird ein Pixel nur neu gezeichnet, wenn der Pixel näher zur Kamera ist!

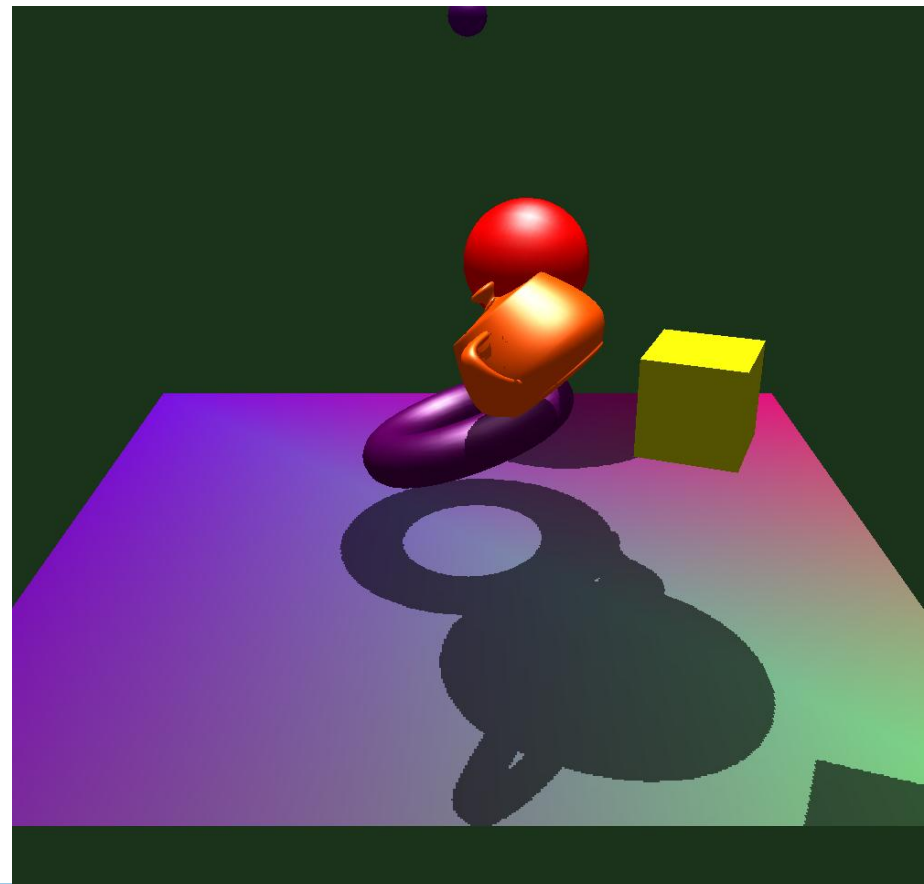
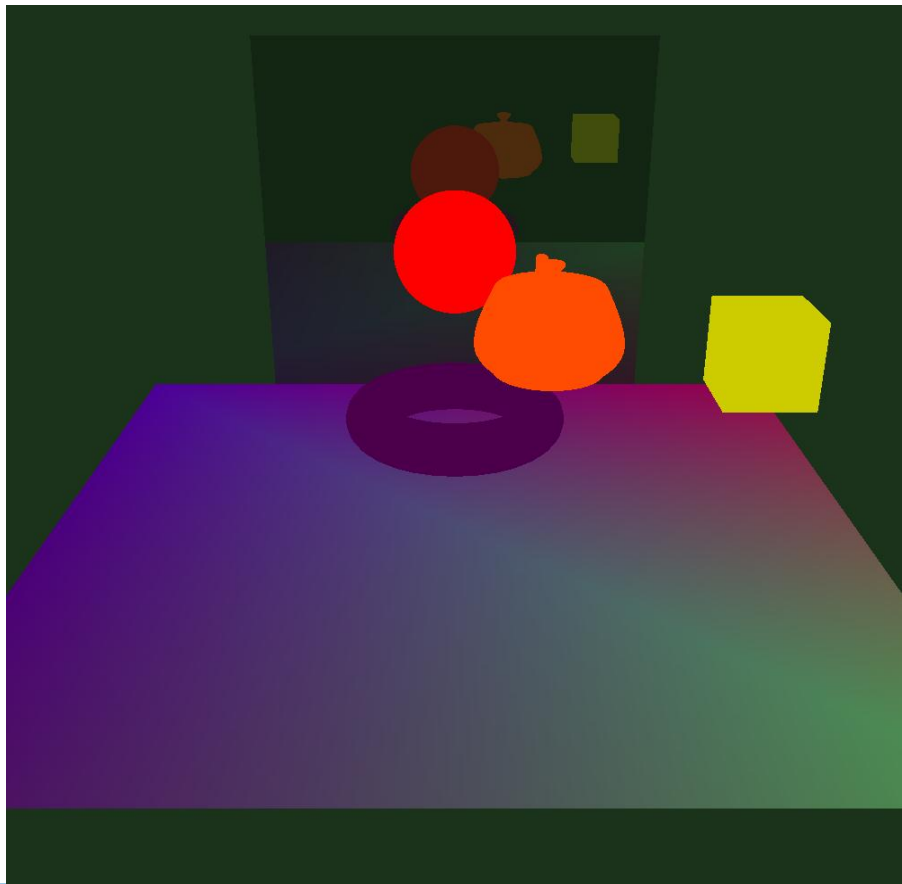
```
glColor3d(0.0,0.0,1.0); // Blau
glutSolidTeapot(10.0);
glTranslate(0.0,0.0,20.0); // 20 nach hinten
glColor3d(1.0,0.0,0.0); // Rot
glutSolidTeapot(10.0);
```

- Ohne Tiefenpuffer werden die Objekte einfach übereinander gelegt.

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(TestType);
```

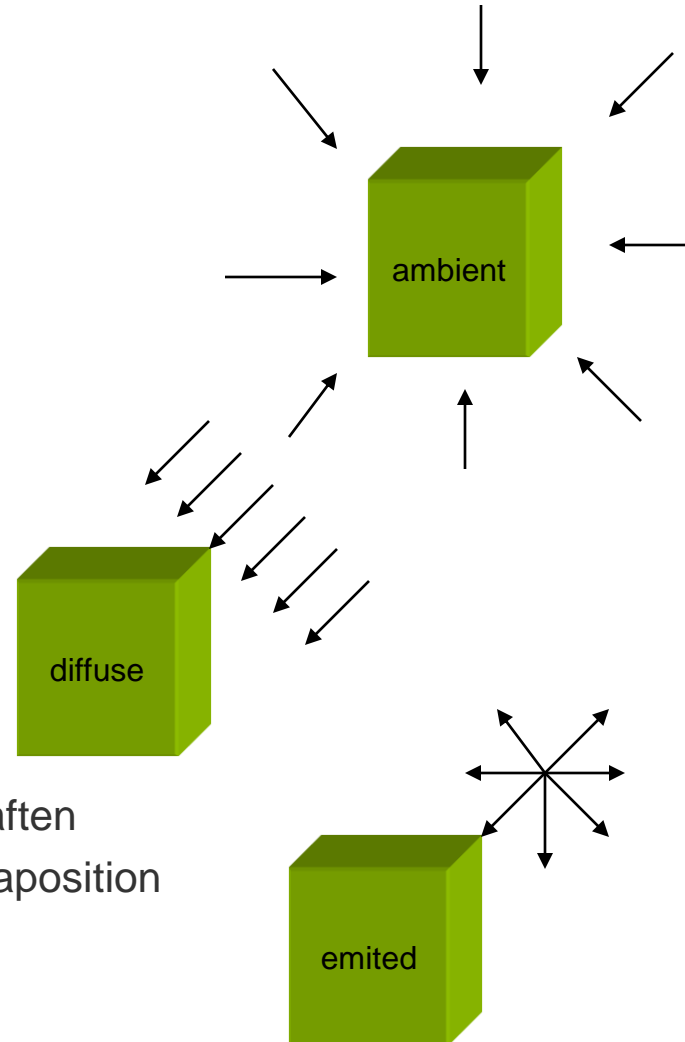
- TestType: GL_LESS (ist standardgemäß voreingestellt), GL_NEVER, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS.

Es werde Licht ...



Verschieden Arten von Licht

- „Ambient Light“
 - Setzt die Grundhelligkeit einer Szene fest
- „Diffuse Light“
 - gerichtetes Licht
 - Quelle im unendlichen
 - Vergleich: Oberflächennormale - Lichtrichtung
- „Specular Light“
 - Gerichtetes Licht mit speziellen Reflexionseigenschaften
 - Vergleich: Oberflächennormale-Lichtrichtung-Kameraposition
- „Emitted Light“
 - Wird vom Polygon selber ausgestrahlt



Licht 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);
```

Maximale Anzahl der Lichtquellen:

```
int maynumlights = 0;
glGetIntegerv(GL_MAX_LIGHTS, &maynumlights);
```

Zuviele Lichtquellen geht zu lasten der Performance!

Licht in OpenGL

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

Unterscheide zwischen

- gerichtetem Licht und **w=0**
- Positionslichtquelle **w!=0**

```
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 15.0f);  
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.0f);  
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.0f);  
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.0f);  
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction);  
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, spotCutoff);
```

Materialeigenschaften für die Objekte

- Für die Reflektionseigenschaften gibt es zwei Möglichkeiten
 - schnell&einfach:

```
glColorMaterial(GL_FRONT_AND_BACK,  
GL_AMBIENT_AND_DIFFUSE);  
glEnable(GL_COLOR_MATERIAL);
```
 - “langsamer”&differenzierter:

```
glMaterialf(GL_FRONT_AND_BACK , GL_SHININESS, 40);  
glMaterialfv(GL_FRONT_AND_BACK , GL_AMBIENT_AND_DIFFUSE,  
vector_ambi_diff);  
glMaterialfv(GL_FRONT_AND_BACK , GL_SPECULAR,  
vector_spec);
```

Displaylisten

- Statische Objekte, die sich zwischen `glBegin()` und `glEnd()` beim erneuten Aufruf nicht ändern, kann man auf der Graphikkarte zwischenspeichern
- Weniger Belastung für den Bus → Schnellere Ausführung
- Besonders lohnenswert bei großen Objekten

Vorgehensweise

- **Leere Liste erzeugen**

```
GLuint model;  
model = glGenLists(2); // Erzeugt zwei Listen und übergibt  
                        // deren Namen an model
```

- **Liste befüllen**

```
glNewList(model, GL_COMPILE);  
    // glBegin und glEnd() u.s.w.  
glEndList();
```

- **Liste anzeigen**

```
glCallList( model );
```

- **Zweite Liste benutzen**

```
glNewList(model+1, GL_COMPILE); ...  
glCallList( model+1 );
```

- **Überprüfen, ob es die Liste n gibt**

```
GLboolean glIsList( model + n );
```

OpenGL Ist kein GUI Toolkit

- OpenGL bieten keine Funktionalität an zu generieren und managen von GUI Fenstern
- Der Setup eines validen OpenGL Render Kontexts wird dem Programmierer überlassen
- Fenster und OpenGL Render Kontext können mit verschiedenen GUI Toolkits erstellt werden, z.B. GLUT, Qt, FLTK, MFC,...
- GLUT – OpenGL Utility Library: Eine plattformunabhängige Bibliothek zum generieren und managen eines simplen Fensters mit OpenGL Kontext.
 - Events wie Maus und Tastatur können auch gehandhabt werden.
- Websites <http://www.opengl.org/resources/libraries/glut/>
, <http://opengl.sourceforge.net/> , <http://www.xmission.com/~nate/glut.html>

Fenstermanagement mit GLUT oder Qt

- GLWidget mit Qt

ODER

- GLUT Bibliothek einbinden (glut.h, glut32.lib aus dem GLUT Verzeichnis)
 - ermöglicht, ein Fenster mit GL Kontext unter verschiedenen Betriebssystemen zu erzeugen
 - ermöglicht, Tastatur, Maus und Keyboardeingaben zu behandeln
 - Download und Infos: <http://www.opengl.org/resources/libraries/glut/>

Setup einer OpenGL App mit GLUT

- Downloaded des GLUT source-code oder der vorkompilierten platform spezifischen Bibliotheken (für Anfänger empfohlen)
- Inkludieren der header files glut.h und glu.h
- Hinzufügen der GLUT und GLU library als linker inputs (glut32.lib, glu32.lib)
- Häufigster Fehler beim Setup von GLUT GLU: Paths zu header und lib Dateien wurden in den MS Projekt Einstellungen nicht hinzugefügt

Download <http://www.opengl.org/resources/libraries/glut/>
, <http://opengl.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 }
93
```

Hilfe?!

- Delphi Wiki (<http://www.delphigl.com/>), ist zwar nicht C++, sondern Delphi, aber OpenGL ist sprachunabhängig!
- NeHe Online Tutorials samt Code (<http://nehe.gamedev.net/>) oder dessen deutsche Übersetzung (http://www.joachimrohde.com/cms/xoops/modules/articles/index.php?ca_t_id=1)
- Originalreferenz von SGI (<http://www.opengl.org/>)
- Das RedBook Version 1.1 online (http://www.opengl.org/documentation/red_book/)
- GLUT Spezifikation (<http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>)
- OpenGL Interactive Tutorials (<http://www.xmission.com/~nate/tutors.html>)