Introduction to OpenGL

What is Open GL?

- OpenGL = Open Graphics Library
- Hardware abstraction layer through its Application Programmer Interface (API)
- Provides low level, platform independent graphics
- Specifically designed for efficient processing in 3D
- It does not have:
  - High level modeling constructs
  - Windowing facilities
  - Input event handling

OpenGL Basics

- OpenGL core libraries:
  - GL (Graphics Library)
  - GLU (Graphics Library Utilities)

```
#inlcude <GL/gl.h>
```

- Basic Syntax
  - Function names prefixed with gl (e.g. `glClear`)
  - Symbolic constants begin with GL_ (e.g. `GL_RGB`)
  - Special built-in data types (`GLbyte, GLfloat, GLint`...)

GLUT

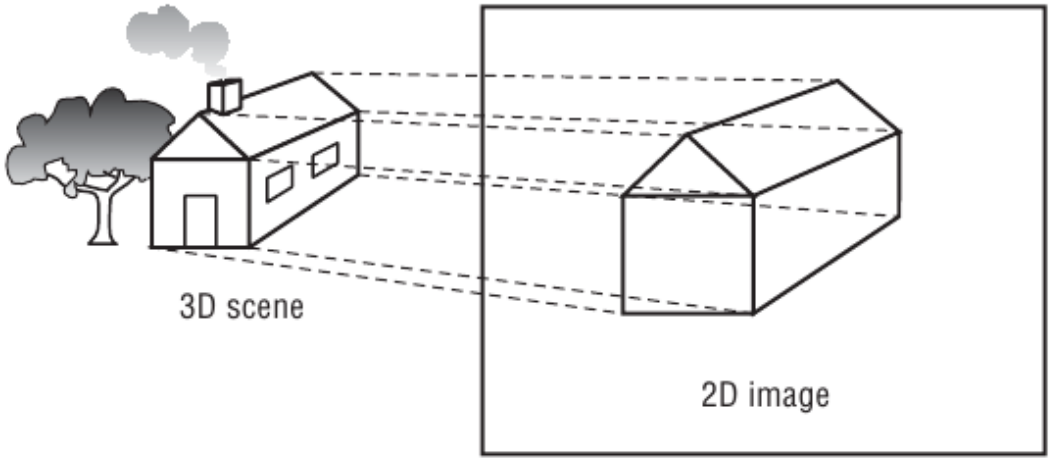
- GLUT = GL Utility Toolkit

```
#inlcude <GL/glut.h>
```

- Easy and stable library for showing OpenGL demos
- Creates a window where the genereted stuff is shown
- Initialized an OpenGL context
- Many callback-function e.g. to handle keyboard or mouse input
- Most OpenGL demos and code on the web use GLUT
- Common Function Prefix: glut (e.g. `glutInit`)

Open GL Architecture

- The Screen is flat but the world is 3D !
  - Don't set the color of each Pixel on the screen manually!
  - Specify the content in 3D space
  - Set up some other parameters
  - OpenGL does the rest



OpenGL Architecture

- OpenGL is a **State Machine**
- Hardware pipeline from the 3D geometry to the final image
- Many things affect how the scene is drawn e.g:
  - What is the Color of an object or the background?
  - From which perspective should the scene be drawn?
  - ...
- Tell it OpenGL with State Variables
  - Global variables which are always valid
  - Remain in effect until they are changed

OpenGL Architecture

- OpenGL offers special functions to change state variables

Examples:

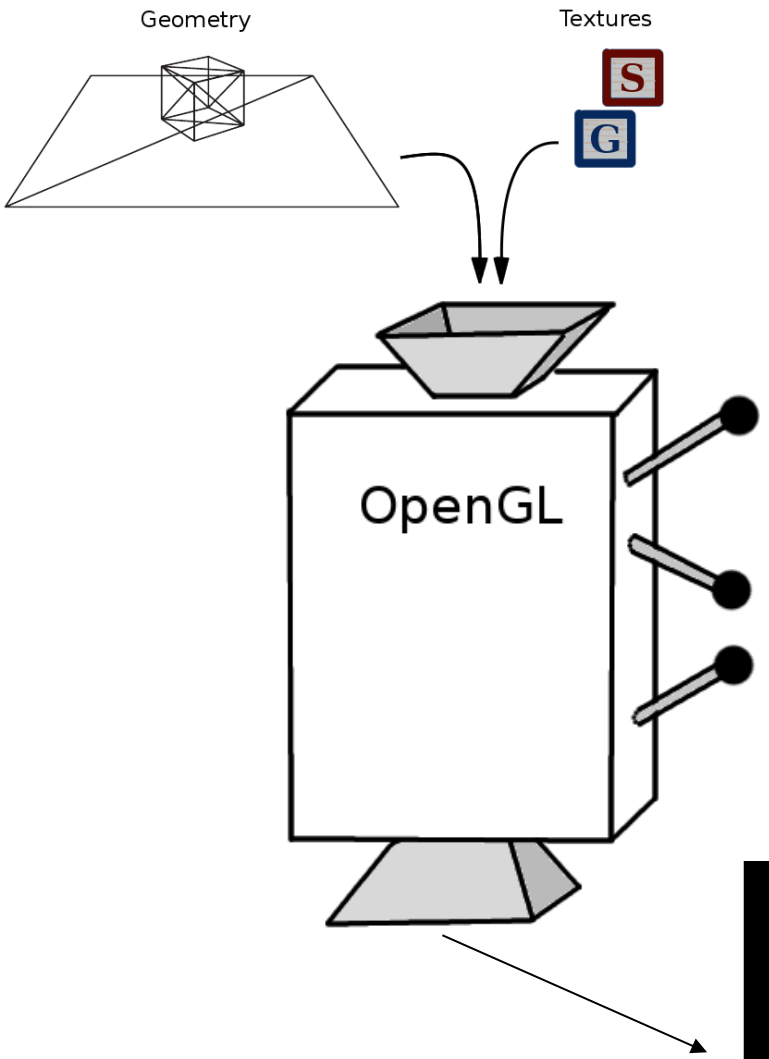
```
glEnable(GL_DEPTH_TEST) // switch on z-test
glDisable(GL_LIGHTING)  // switch off lighting
```

- State variables have default values

Examples:

```
GL_CURRENT_COLOR    :    (1,1,1,1)
GL_BLEND            :    GL_FALSE
GL_FRONT_FACE       :    GL_CCW
```

OpenGL - fixed function pipeline



Camera Transformations

Ligting Parameters

Texture Mapping

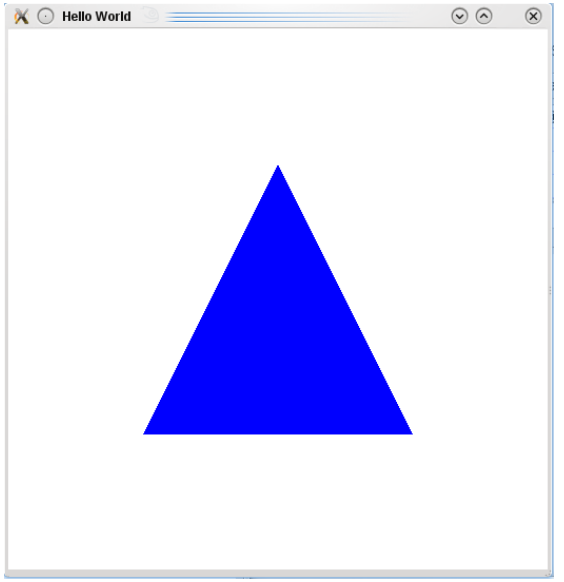
.....

Getting a Window and an OpenGL Context

```
#include <GL/glut.h>

int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitWindowSize(512, 512);
      glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
      glutCreateWindow("Hello World");

      glutMainLoop();
}
```

How do we create content ?

```
#include <GL/glut.h>

void display() {

   draw a blue Triangle!

}

int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitWindowSize(512, 512);
      glutInitDisplayMode(GLUT_RGB);
      glutCreateWindow("Hello World");

      glutDisplayFunc(display);

      glutMainLoop();
}
```



An example display-function

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,1.0);

        glBegin(GL_TRIANGLES);
                glVertex3f(-0.5,-0.5, -1);
                glVertex3f(0.5,-0.5,-1);
                glVertex3f(0,0.5,-1);
        glEnd();

        glFlush();
}
```

- Clear the Framebuffer before you draw the scene

- Takes the color in the state variable GL_CLEAR_COLOR which is set by glClearColor (Default is black)

- Additional Buffer can be cleared with a logical or:

e.g **glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);**

An example display-function

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,1.0);

        glBegin(GL_TRIANGLES);
                glVertex3f(-0.5,-0.5, -1);
                glVertex3f(0.5,-0.5,-1);
                glVertex3f(0,0.5,-1);
        glEnd();

        glFlush();
}
```

- Set the current color to blu

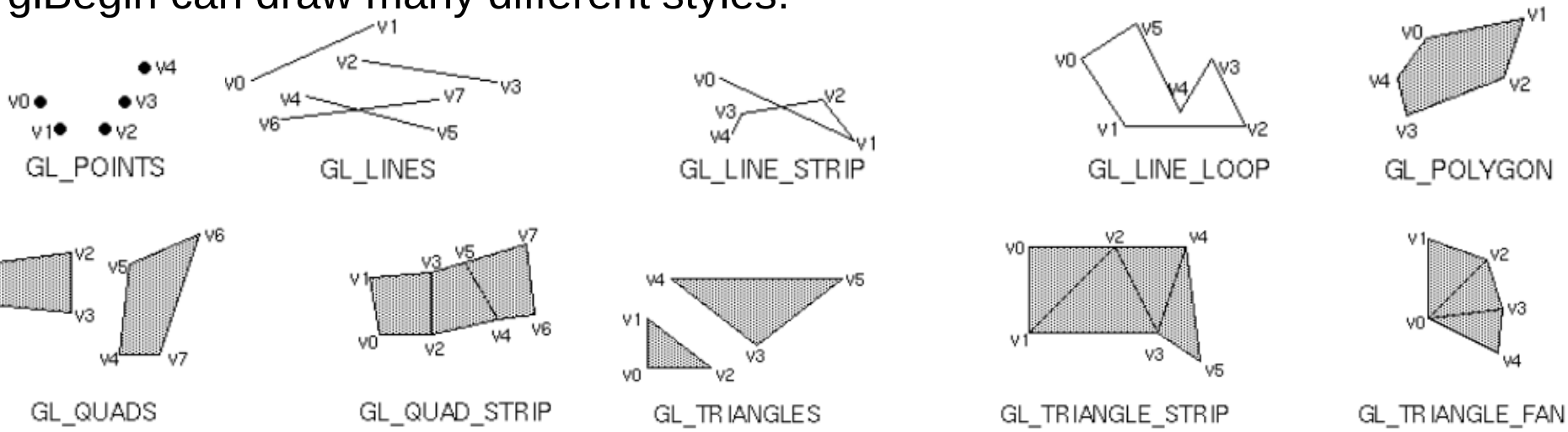
- All specified vertices from here on will get this attribute

An example display-function

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,1.0);

        glBegin(GL_TRIANGLES);
                glVertex3f(-0.5,-0.5, -1);
                glVertex3f(0.5,-0.5,-1);
                glVertex3f(0,0.5,-1);
        glEnd();

        glFlush();
}
```

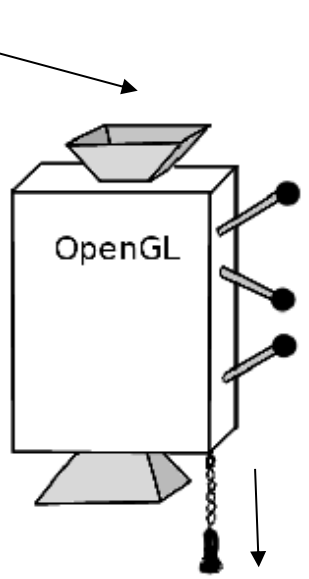
- glBegin starts the “immediate mode” to send down Vertex data down the pipeline.
- glEnd stops it.
- In between, glVertex3f(x,y,z) sepcifies a point in 3D space
- if you draw polygons, the vertices should be ordered in counterclockise order to indicate the front face.

- glBegin can draw many different styles:



An example display-function

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,1.0);
        glNormal3f(0,0,1);

        glBegin(GL_TRIANGLES);
                glVertex3f(-0.5,-0.5, -1);
                glVertex3f(0.5,-0.5,-1);
                glVertex3f(0,0.5,-1);
        glEnd();

        glFlush();
}
```

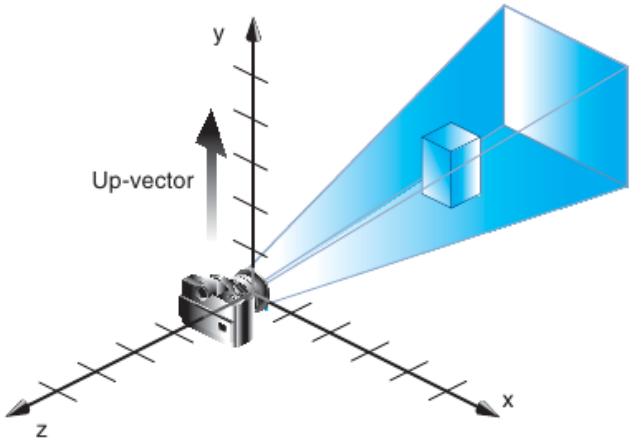


- Forces execution of GL commands

- If you use double buffering, you have to call **glutSwapBuffers()** instead

Viewing

- Position and direction of the viewer
- Influences the **Modelview Matrix**




```
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
```

`eyeX,eyeY,eyeZ`: Position of the camera in world coordinates.

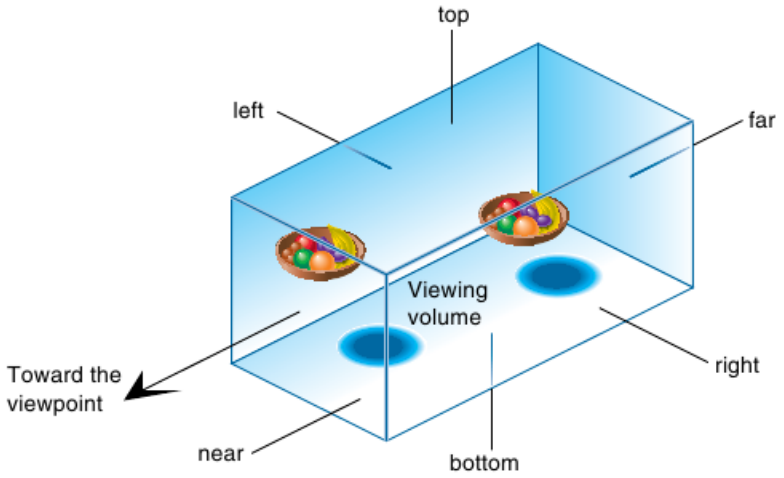
`centerX,centerY,centerZ`: Point in world coordinates, to which the camera is pointing

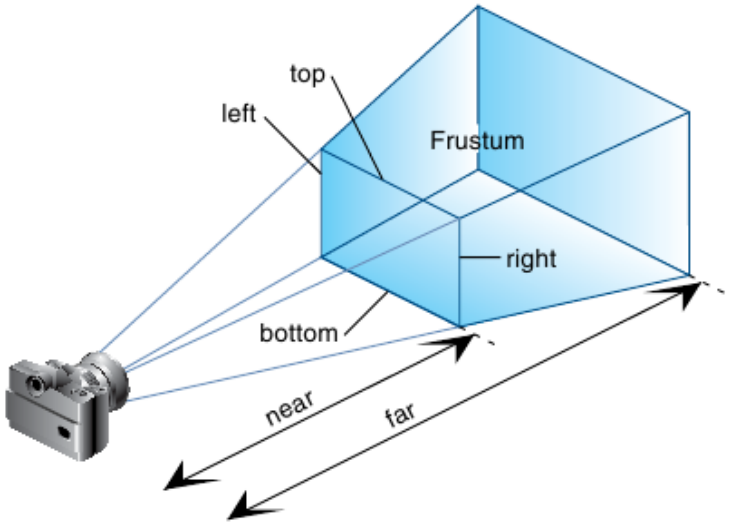
-> view direction = center - eye

`upX,upY,upZ`: Vector, pointing upwards from the viewer.

Viewing

- Viewing Volume
  - Determines the space which is projected on the screen
  - Orthographic or perspective projection
  - Influences the **Projection Matrix**



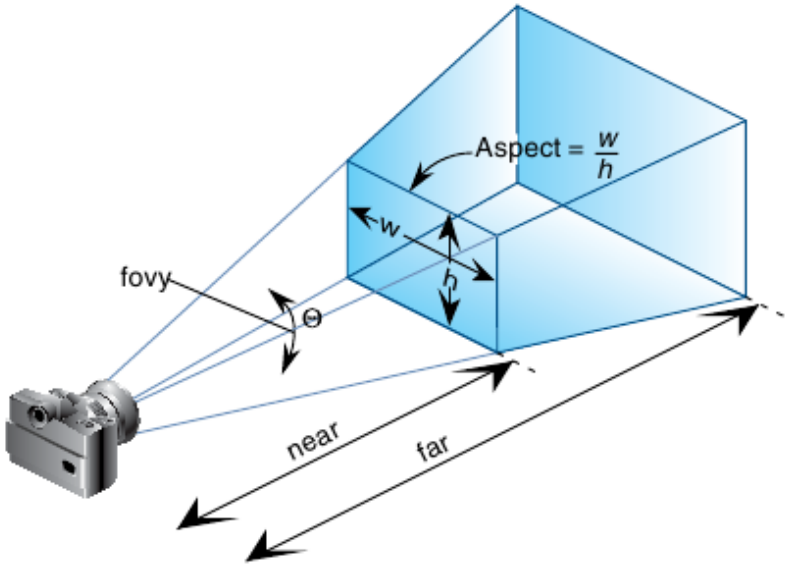


`glOrtho(left, right, bottom, top, near, far)`

`glFrustum(left, right, bottom, top, near, far)`

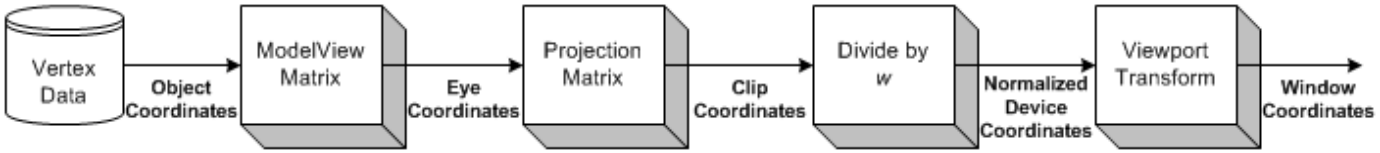
Viewing

- Alternative (more intuitive) way to specify a view frustum




```
gluPerspective(fovy, (GLfloat) aspect , near, far)
```

Transformations



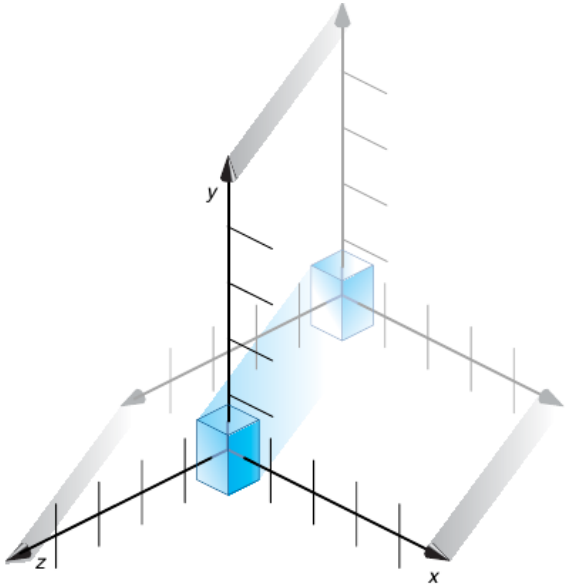
Modelview Matrix
positions the objects / camera in the scene

Projection Matrix
determines the viewing volume and how things are projected on the screen

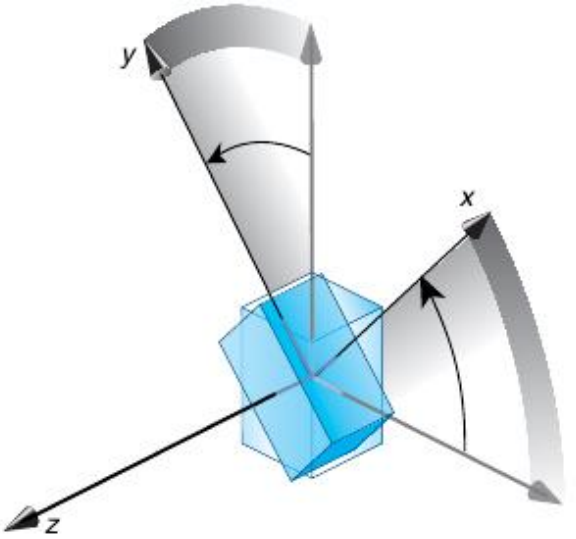
- glMatrixMode specifies which matrix is influenced by subsequent operations (GL_PROJECTION or GL_MODELVIEW)
- glLoadIdentity sets the current matrix to the identity matrix
- A transformation operation produces the according matrix and multiplies it to the current matrix from the right.

Transformations

- Examples



`glTranslatef(x, y, z);`



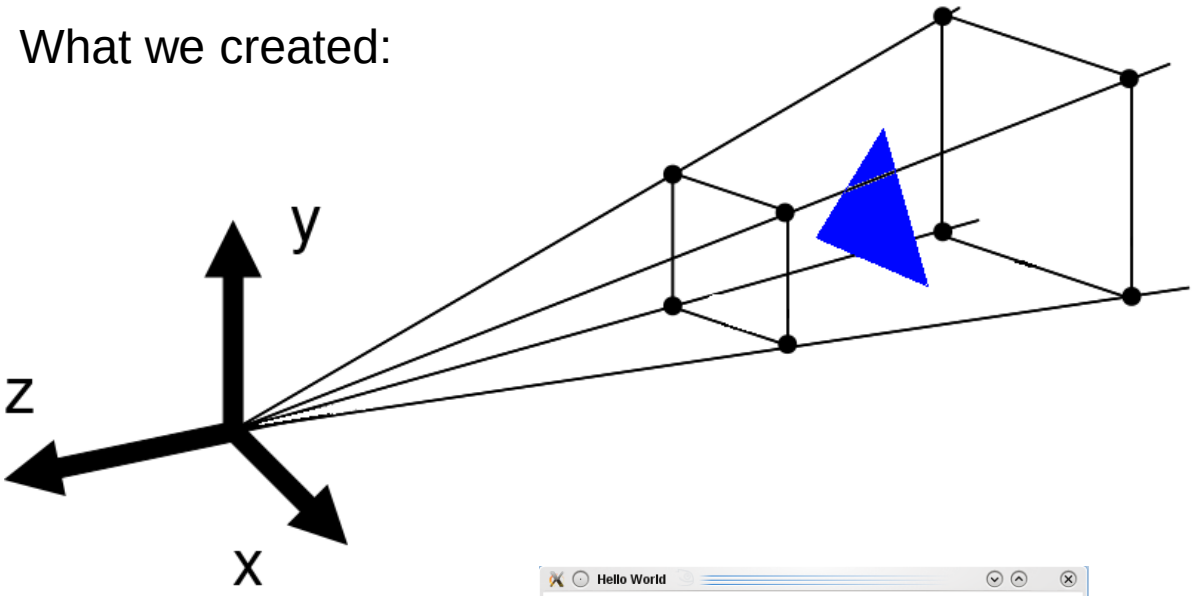
`glRotatef(angle, x, y, z);`

- The center of rotation is always the origin
- Transformations are not commutative!

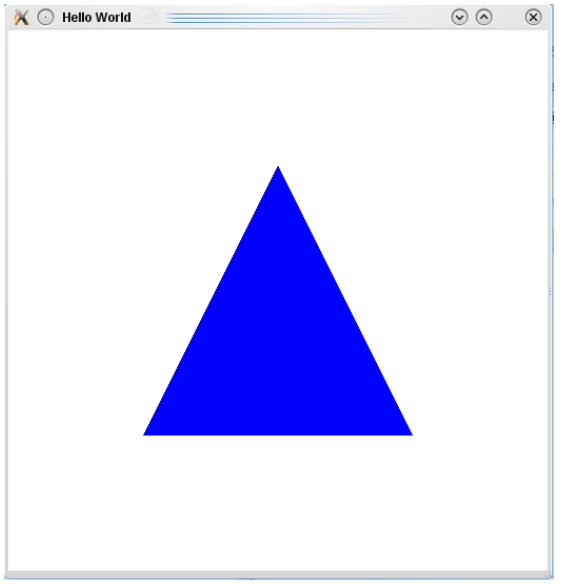
Example

```
#include <GL/glut.h>

void init_openGL() {

          glMatrixMode(GL_PROJECTION);
          glLoadIdentity();
          glFrustum(-1,1,-1,1,1,5);

          glMatrixMode(GL_MODELVIEW);
          glLoadIdentity();
          gluLookAt(0,0,0,0,0,-1,0,1,0);
}

void display()
{
          glClear(GL_COLOR_BUFFER_BIT);

          glBegin(GL_TRIANGLES);
                    glColor3f(0,0,1);
                    glVertex3f(-1,-1,-2);
                    glVertex3f( 1,-1,-2);
                    glVertex3f( 0, 1,-2);
          glEnd();
          glFlush();

}


int main(int argc, char **argv)
{
          glutInit(&argc, argv);
          glutInitWindowSize(512, 512);
          glutInitDisplayMode(GLUT_RGB);
          glutCreateWindow("Hello World");

          init_openGL();
          glutDisplayFunc(display);
          glutMainLoop();
          return 0;
}
```

What we created:



What we see:



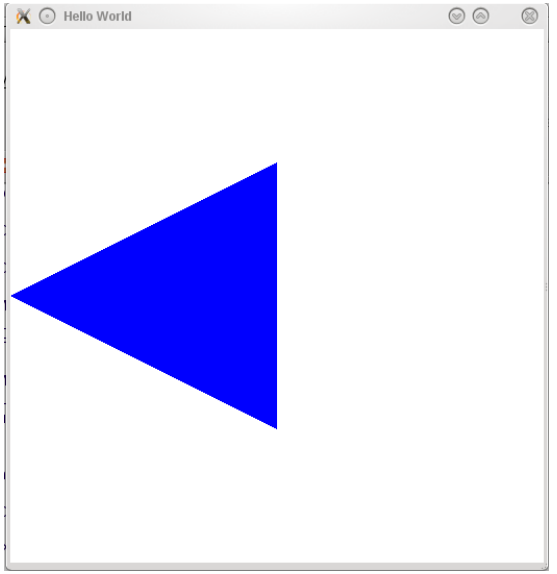
Transformations

```
#include <GL/glut.h>

void init_openGL() {

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glFrustum(-1,1,-1,1,1,5);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0,0,0,0,0,-1,0,1,0);
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glPushMatrix();
        glTranslatef(0.0, 1.0, 0.0);
        glRotatef(90,0,0,1);


        glBegin(GL_TRIANGLES);
                glColor3f(0,0,1);
                glVertex3f(-1,-1, -2);
                glVertex3f(1,-1,-2);
                glVertex3f(0,1,-2);
        glEnd();
        glPopMatrix();
        glFlush();

}                          ...
```

- OpenGL provides many functions to perform Transformations (glRotate,glTranslate,glScale...)
- Internally, a transformation matrix is built and multiplied to the active matrix (which should be the Modelview Matrix) from the right.

Careful with the order of Transformations!

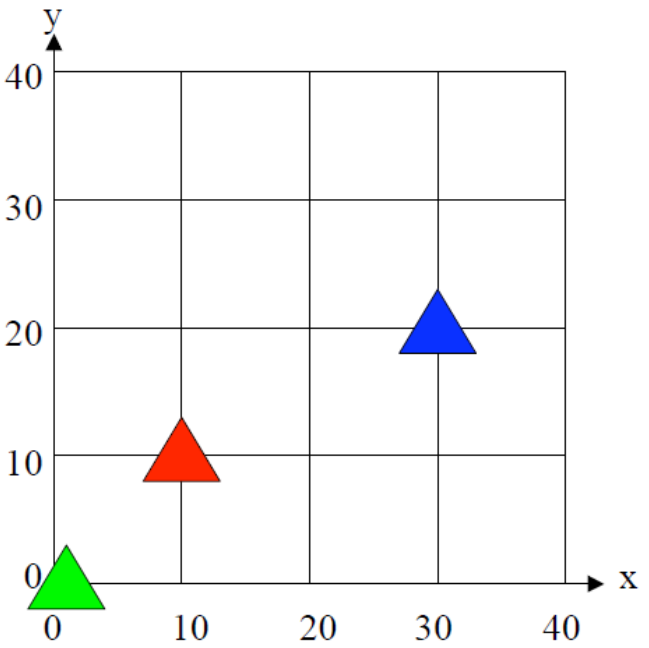
- First 90 degrre Rotation around the z-axis
- Then Translation of 1.0 in y-direction



The Matrix Stack

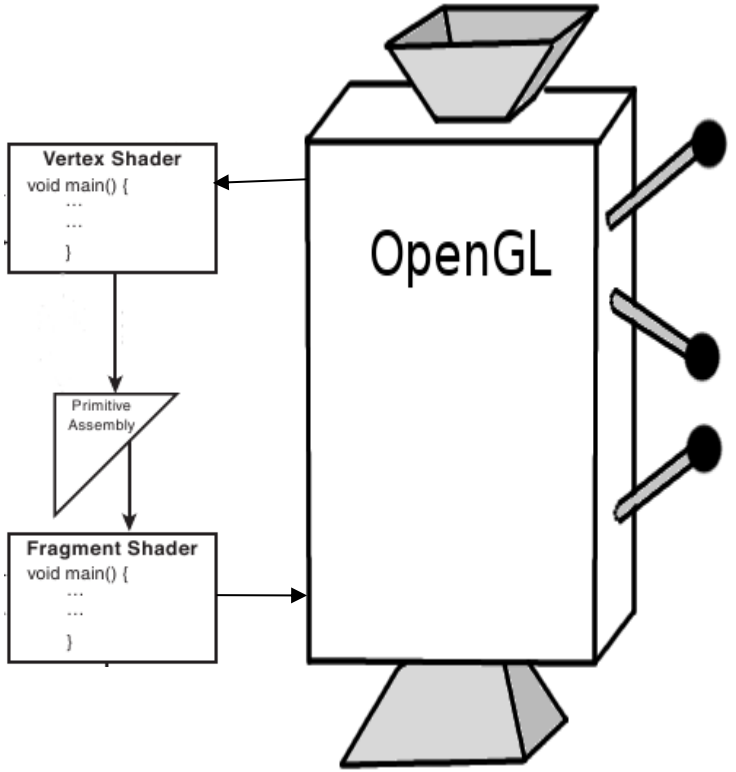
- **glPushMatrix()**: Make a copy of the current matrix and put it on top of the stack. (save the current Position)
- **glPopMatrix()**: Remove the top matrix from the stack

```
glPushMatrix();
glTranslatef(10,10,0);
drawRedTriangle();
      glPushMatrix();
      glTranslatef(20,10,0);
      DrawBlueTriangle();
      glPopMatrix();
glPopMatrix();
DrawGreenTriangle();
```



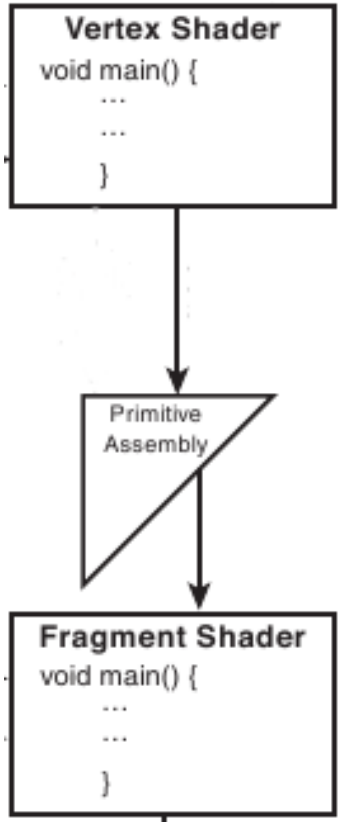
Shader

- Fixed function pipeline is convenient, but offers only limited quality
- Graphic cards become more and more programable
- Basic Blocks: Vertex and Fragment Shader
- Shaders are optional in OpenGL 2.x and mandatory since OpenGL 3.x



- You will work with GLSL (GL Shading Language)

Shader



- Vertex Shader
  - Executed for each vertex sent down the pipeline
  - Performs transformations or other types of math on the vertex
- Fragment Shader
  - Executed for each fragment generated in the rasterizer
  - Outputs the final color value for each pixel on the screen

Vertex Shader

- Input:

Vertex attributes (Read Only)

- `vec4 gl_Color`
- `vec4 gl_Vertex`
- `vec3 gl_Normal`
- ...

```
void main()  {


}
```

to rasterizer

- Output

Vertex Position: (must be set)

- `vec4 gl_Position`

Varying Variables

- `vec4 gl_FrontColor`
- `vec4 gl_TexCoord[]`
- `...`

- Varying variables are interpolated in the rasterization step
- Input and varying variables can also be user defined

Vertex Shader

- Example

```
void main(){

        // Transforming The Vertex

        gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;


        // Add a little red to the color:

        gl_Front_Color += vec4(0.2, 0, 0, 0);


}
```

Fragment Shader

from rasterizer →

- Input:

Varying Variables

- `vec4 gl_Color`
- `vec4 TexCoord[]`
- ...

Special Input Variables

- `vec4 gl_FragCoord`
- `...`

```
void main()  {


}
```

- Output

The Color

- `vec4 gl_FragColor`

Fragment Shader

- Example: Set all even columns to black

```
void main(){


      vec4 resulting_color;

      if (mod(gl_FragCoord.x,2) == 0)  resulting_color = vec4(0,0,0,0);

      else resulting_color = gl_color;

      gl_FragColor = resulting_color;

}
```

Creating, compiling and linking shader objects

```
GLenum my_program;
GLenum my_vertex_shader;
GLenum my_fragment_shader;

// Create Shader And Program Objects
my_program = glCreateProgramObjectARB();
my_vertex_shader = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
my_fragment_shader = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);

// Load Shader Sources
glShaderSourceARB(my_vertex_shader, 1, &my_vertex_shader_source,NULL);
glShaderSourceARB(my_fragment_shader, 1, &my_fragment_shader_source, NULL);

 // Compile The Shaders
glCompileShaderARB(my_vertex_shader);
glCompileShaderARB(my_fragment_shader);

// Attach The Shaders Objects To The Program Object
glAttachObjectARB(my_program, my_vertex_shader);
glAttachObjectARB(my_program, my_fragment_shader);

glLinkProgramARB(my_program);     // Link The Program Object
glUseProgramObjectARB(my_program);   // Use The Program
```

GLUT Callback Funtions

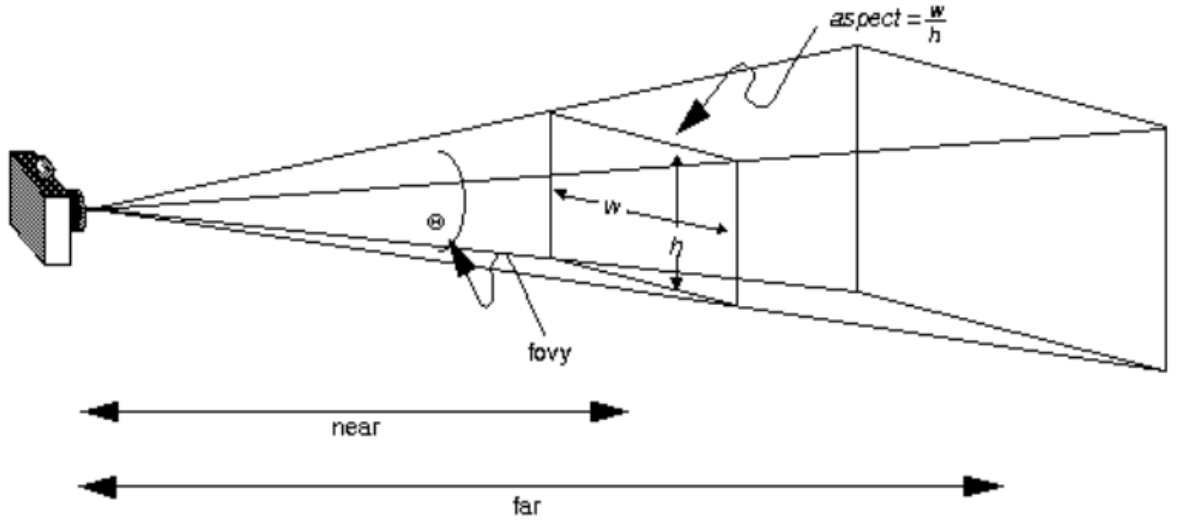
- Assign Callback functions in the main function

```
int main(int argc, char **argv)
{

      glutInit(&argc, argv);
      glutInitWindowSize(1600, 1200);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |GLUT_DEPTH);
      glutCreateWindow("OpenGL Test");

      init_openGL();

      glutDisplayFunc(display);
      glutReshapeFunc(reshape);          //  Window is resized
      glutMouseFunc(onMouseDown);   //  Mouse Button is pressed
      glutMotionFunc(onMouseMove); //  Mouse is moved while button is pressed
      glutKeyboardFunc(keyboard);   //  key is pressed
      glutIdleFunc(onIdle);              //  when nothing else happens

      glutMainLoop(); }
```

glutReshapeFunc

- Typical glutReshapeFunc:

```
void reshape(int w, int h)
{
      glViewport(0,0,(GLsizei) w, (GLsizei) h);

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      float aspect = (float)w/ (float)h;
      gluPerspective(fovy, (GLfloat) aspect , near, far)
}
```



Mouse Functions

- glutMouseFunc

```
void onMouseDown(int button, int state, int x, int y) {

        // button reports the mouse button which was pressed
        // state is either GLUT_UP or  GLUT_DOWN
        // x and y provide the window coordinates of the mouse

}
```

- glutMotionFunc

```
void onMouseMove(int x, int y) {

            // x and y provide the window coordinates of the mouse

}
```

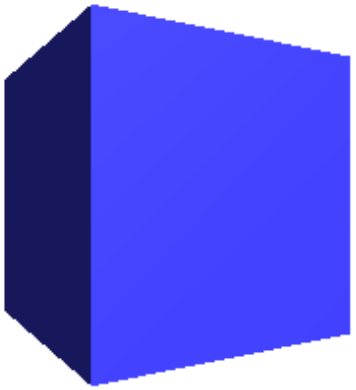
GLUT Objects

- GLUT offers various functions to draw primitive Objects (Cubes, Spheres, Cones...)
- Syntax: `glut[Solid | Wire]Shape`
- Solid objects come with normal vectors (Important for lighting)
- Objects are drawn at the origin, must be positioned with transformations

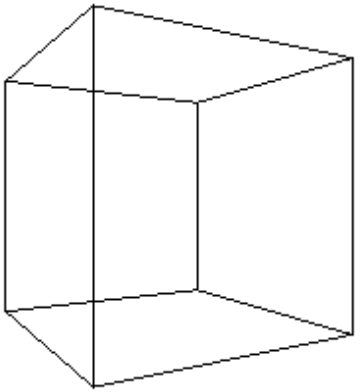
GLUT Objects

- Examples

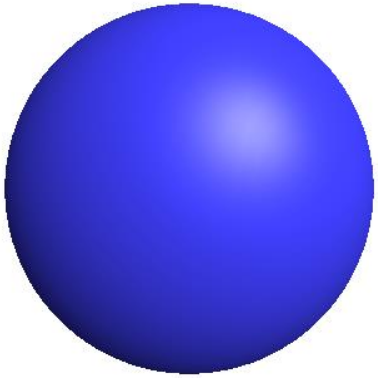
glutSolidCube(GLdouble size)

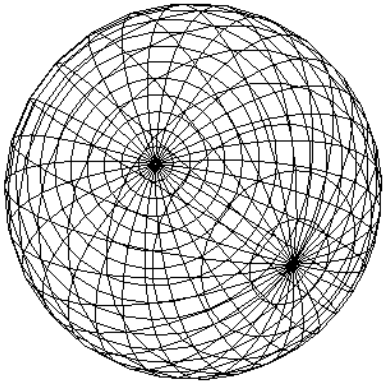


glutWireCube(GLdouble size)



void **glut**^{**Solid**}_{**Wire**}**Sphere**(GLdouble radius, GLint slices, GLint stacks);



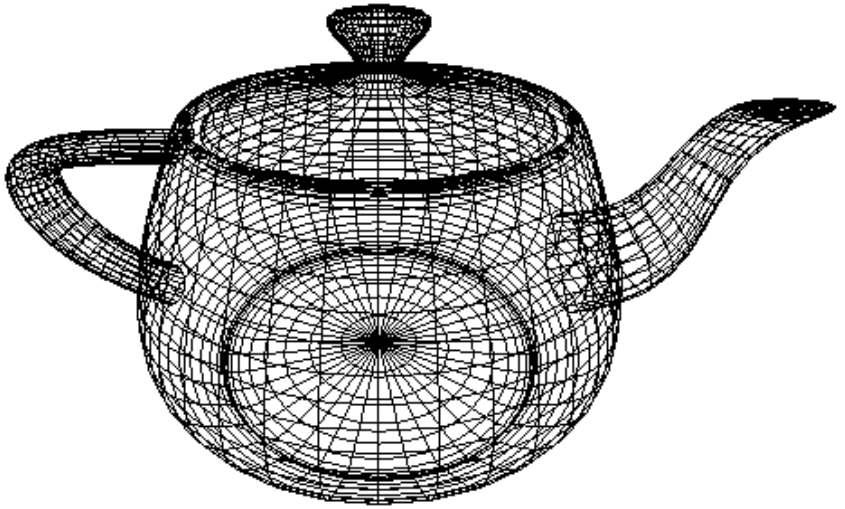


GLUT Objects

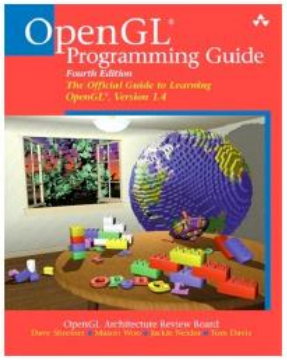
- Examples

glutSolidTeapot(GLdouble size)

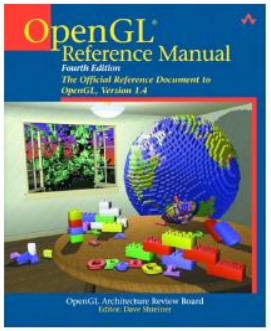
glutWireTeapot(GLdouble size)



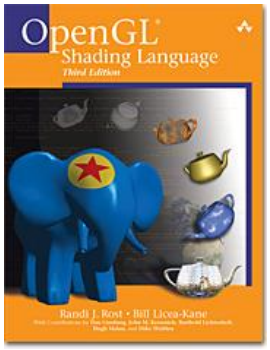
Resources



The Red Book (Programming Guide)
http://www.glprogramming.com/red *(old Version)*



The Blue Book (Function Reference)
http://www.glprogramming.com/blue



The Orange Book (GLSL Guide)

http://opengl-doc.com/Addison.Wesley-OpenGL.Shading1/tindex.htm