Development of Games

Lecture 9

Introduction to OpenGL

(used slides from Ulf Assarsson,
Department of Computer Engineering
Chalmers University of Technology
and Alexey Ignatenko, Moscow State University)

SGI and GL

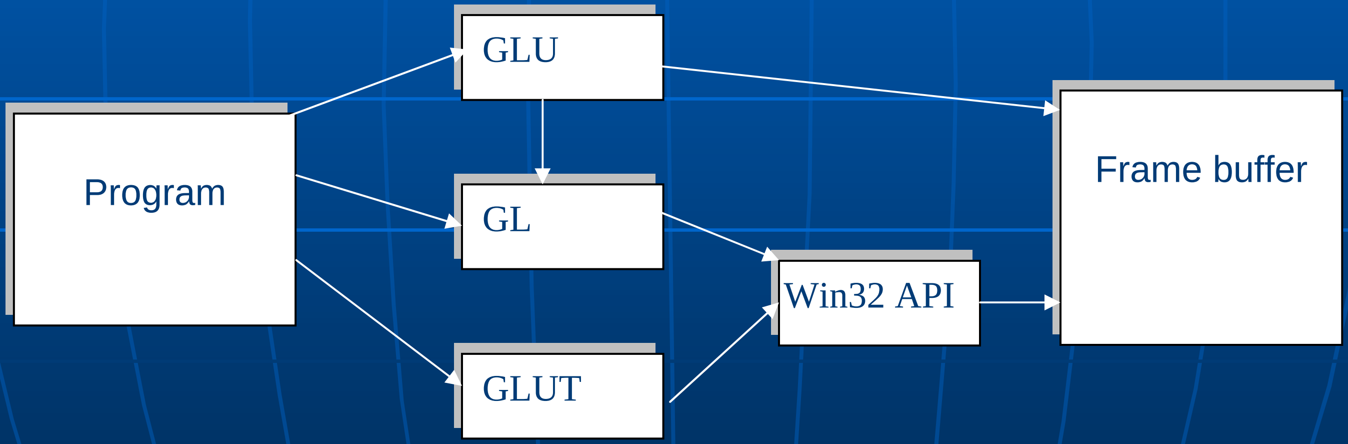
- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

OpenGL

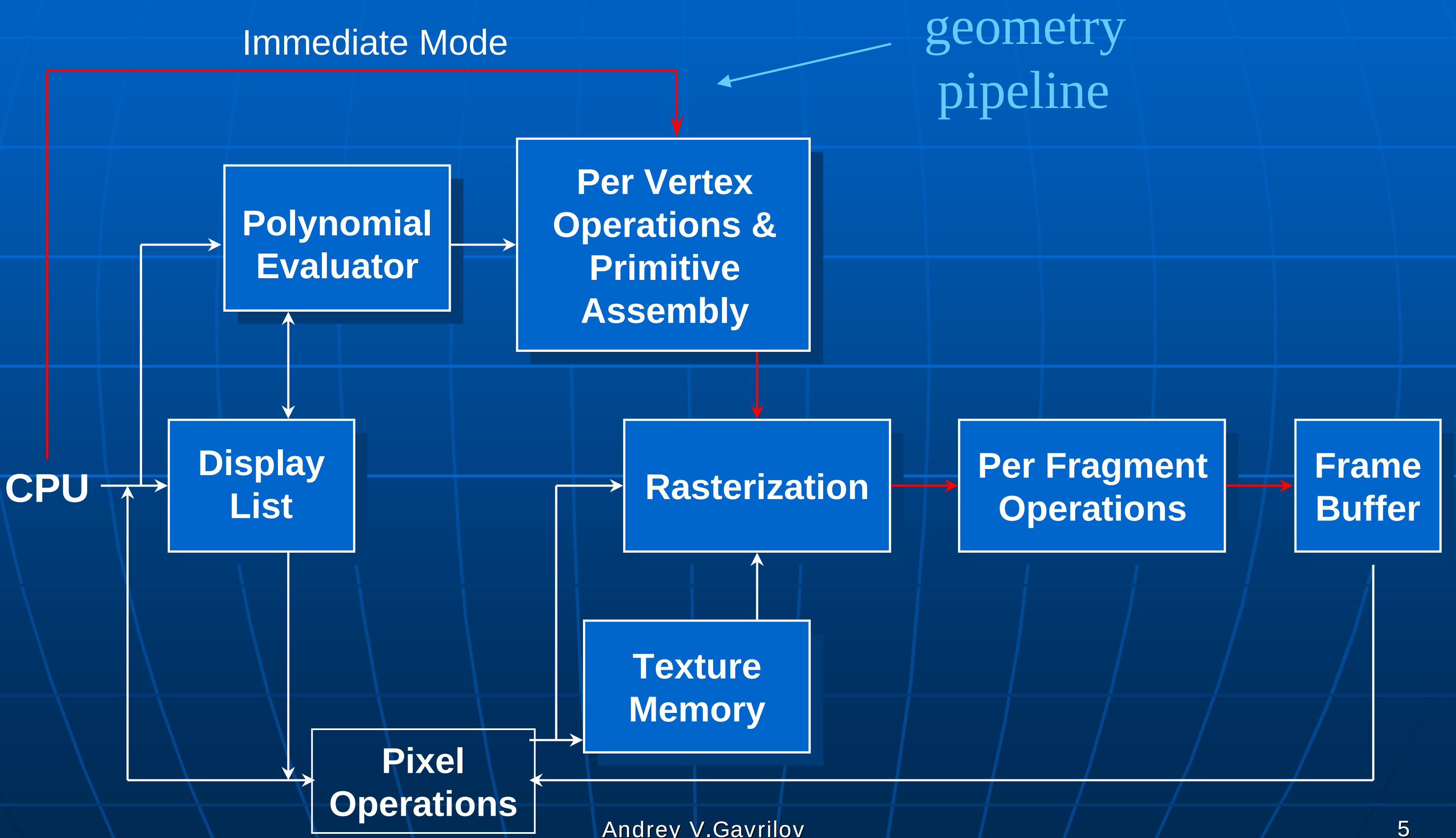
- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies

Architecture of OpenGL

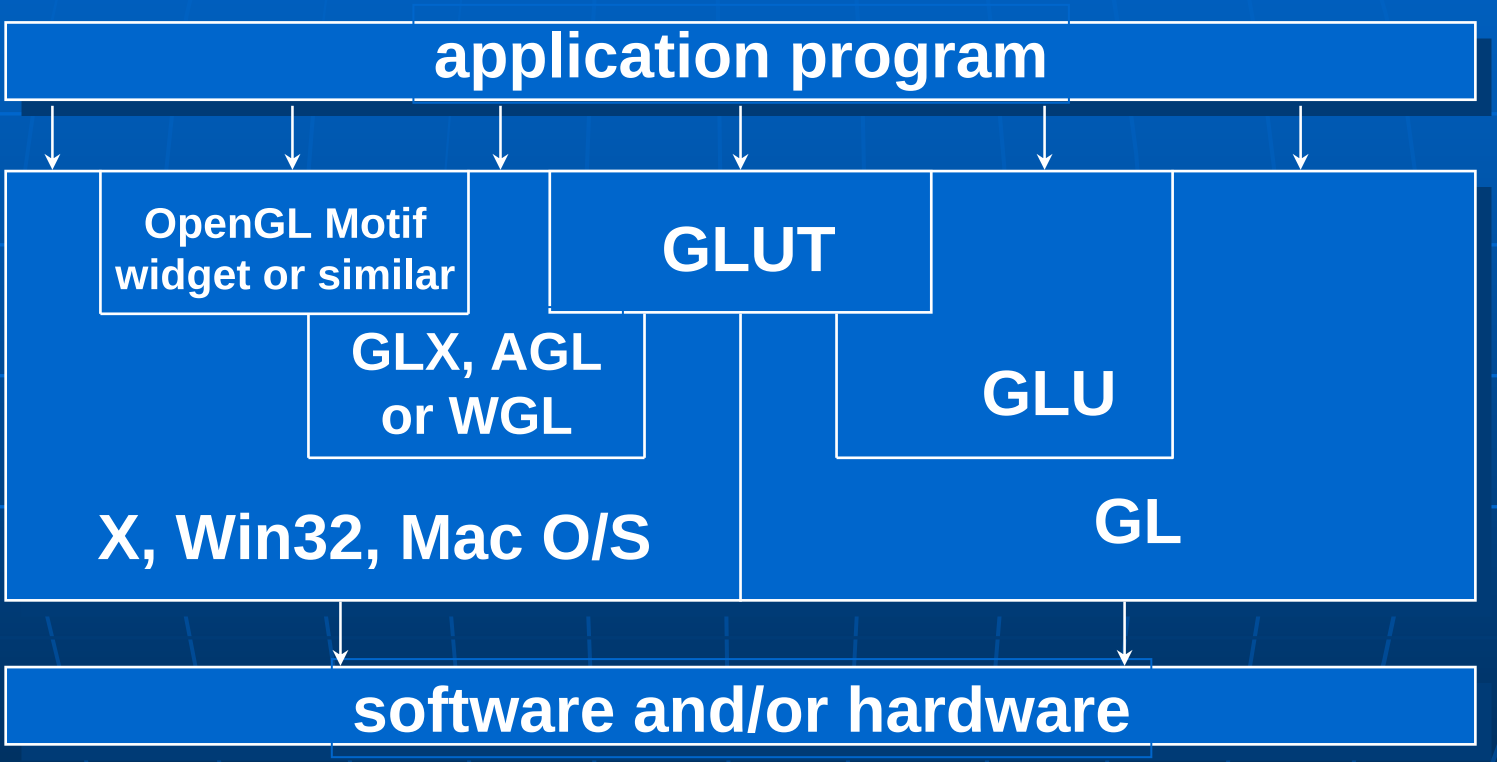
- Set of libraries
  - E.g. for Windows



OpenGL Architecture



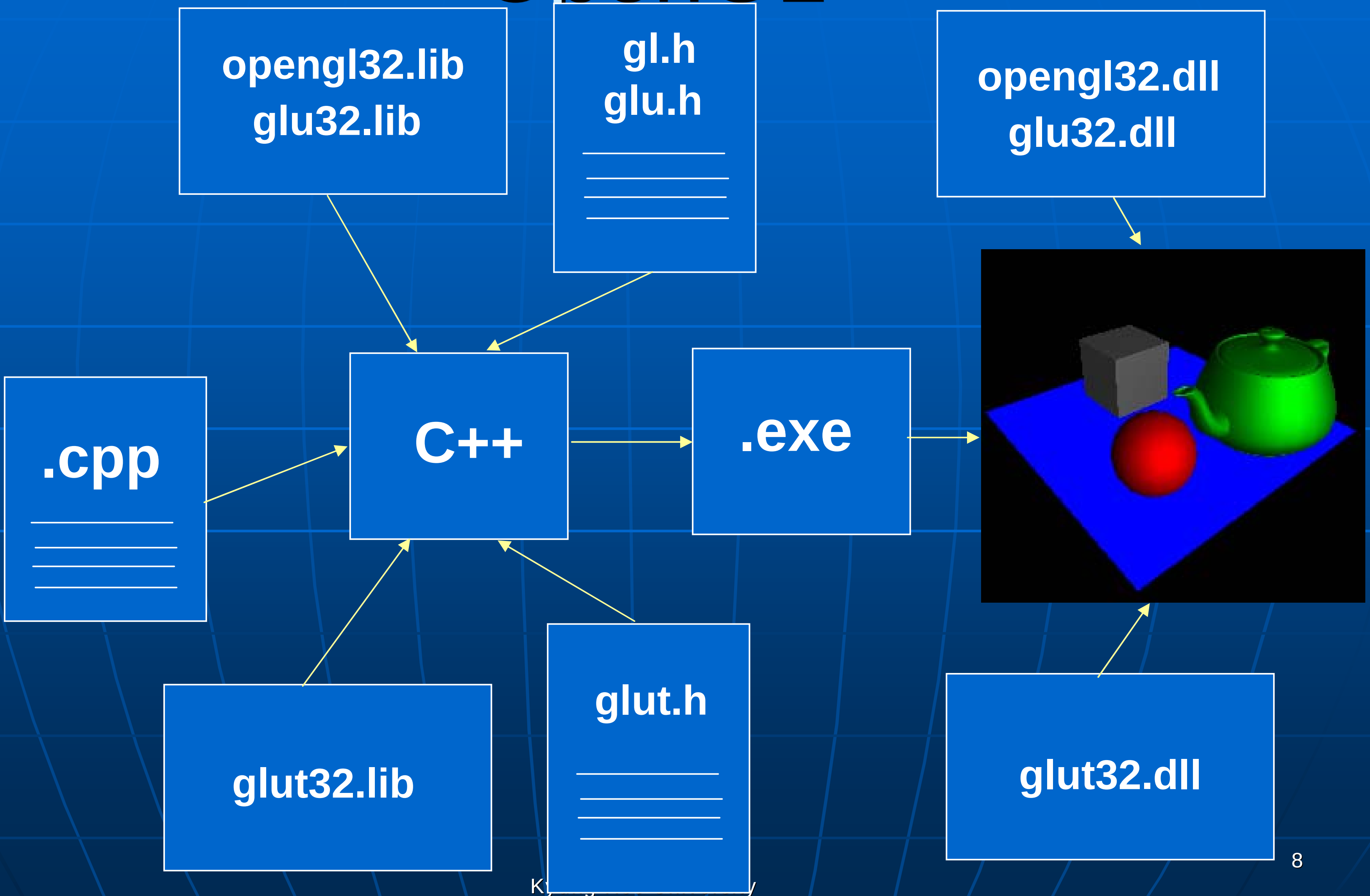
Software Organization



Connecting API's

- AGL, GLX, WGL
  - Links between OpenGL and Windows System
- GLU (OpenGL Utility Library)
  - Part of OpenGL
  - NURBS, tessellators, quadric shapes, etc
- GLUT (OpenGL Utility Toolkit)
  - Removable API
  - Unofficial part of OpenGL

What is needed for working with OpenGL



opengl32.lib
glu32.lib

gl.h
glu.h

opengl32.dll
glu32.dll

.cpp

C++

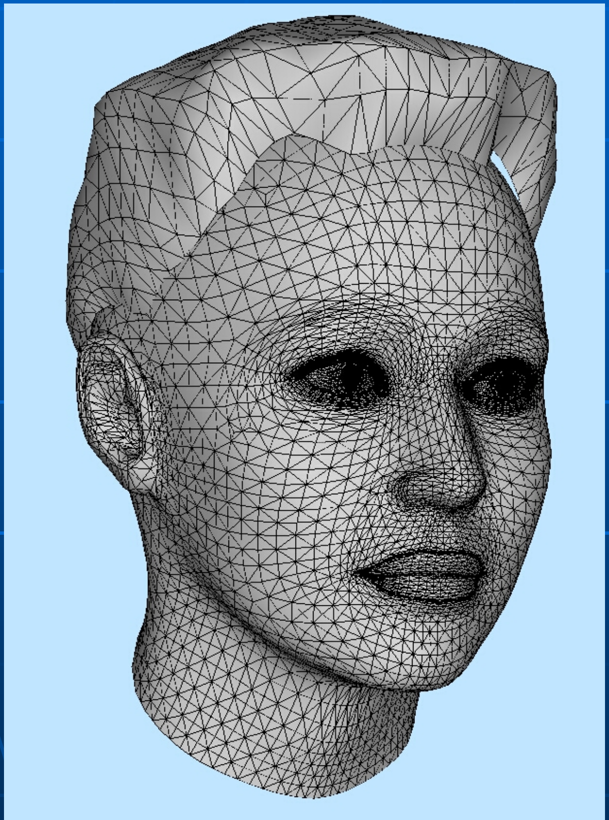
.exe

glut32.lib

glut.h

glut32.dll

Primitives

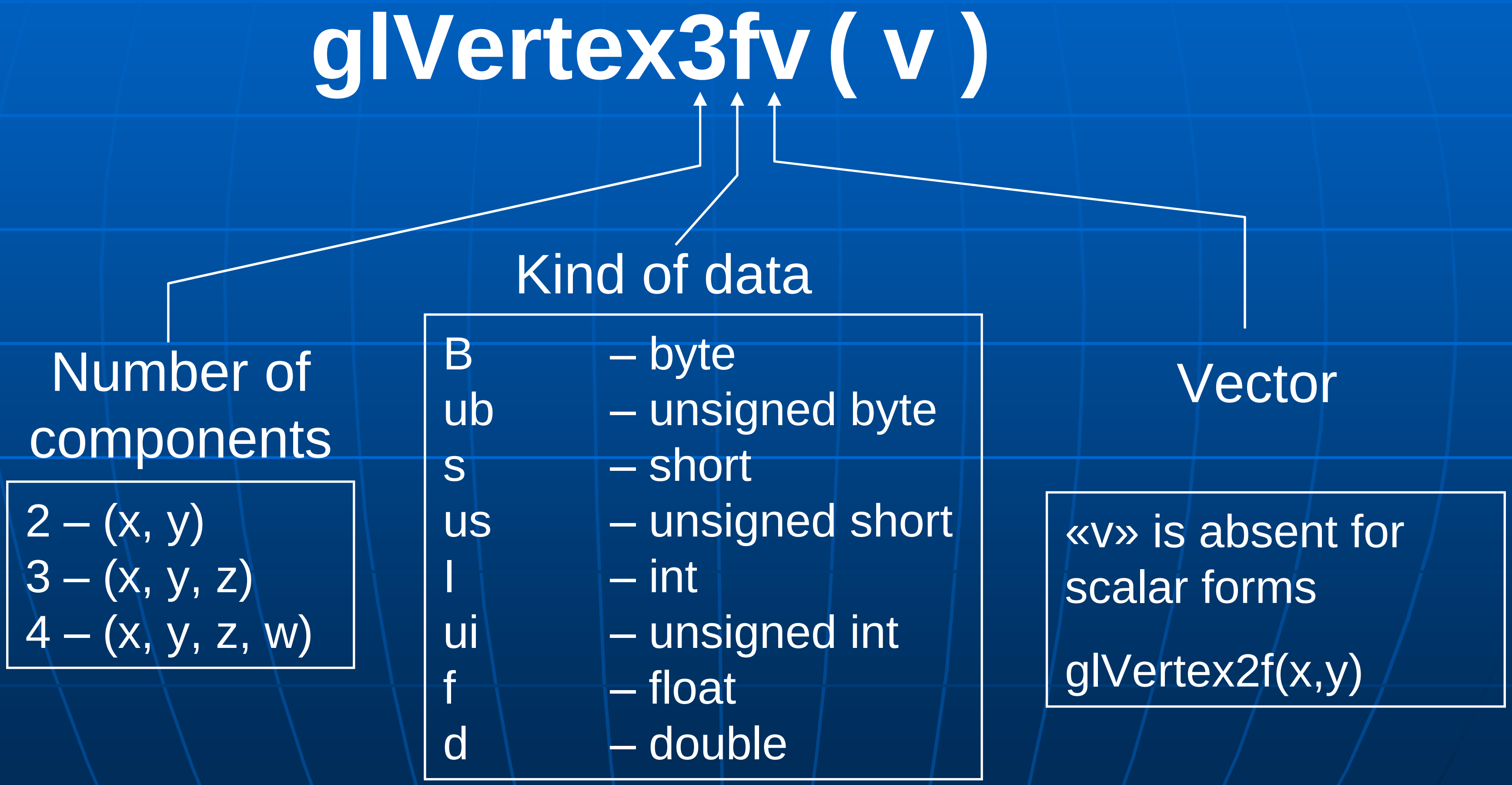


- Polygones
  - Edges
  - Vertex

Commands of OpenGL

- Description of primitives
  - Points, triangles, polygons, vertex and so on
- Description of lighting
  - Position, color and so on
- Definition of attributes
  - Color, material, texture
- Transformations
  - Rotation, translation, camera
- Visualization
  - Control of output on screen

Functions of OpenGL



glVertex3fv (v)

Number of components

2 – (x, y)
3 – (x, y, z)
4 – (x, y, z, w)

Kind of data

B	– byte
ub	– unsigned byte
s	– short
us	– unsigned short
I	– int
ui	– unsigned int
f	– float
d	– double

Vector

«v» is absent for scalar forms

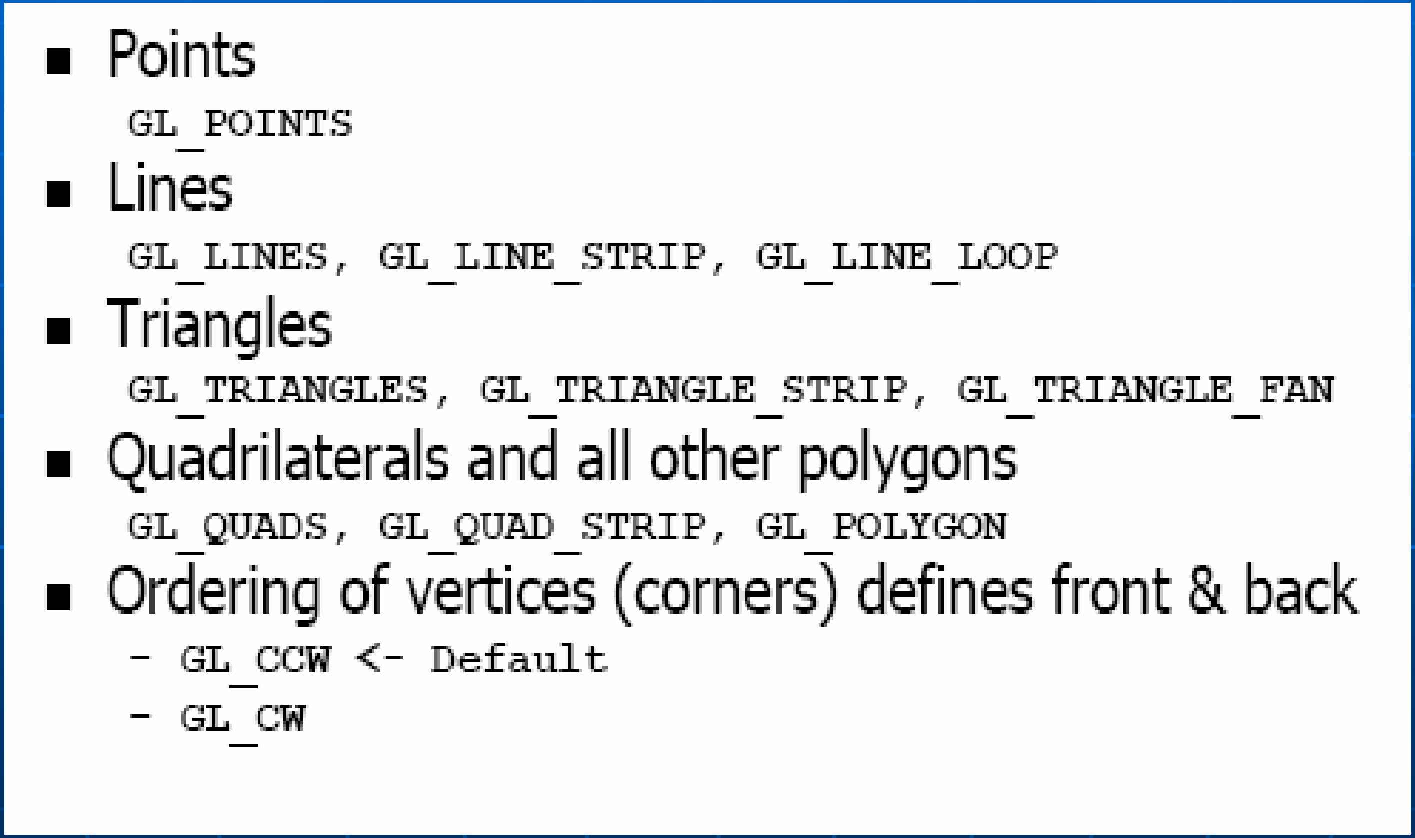
glVertex2f(x,y)

Definition of objects in OpenGL by primitives

```
glBegin( prim_type );

                    glVertex{234}{df}[v]()

glEnd();
```

```
glBegin (GL_POINTS);
   glVertex2f (-0.25, -0.25);
   glColor3f (0.0, 0.0, 1.0);
   glVertex2f (-0.25, 0.25);
   glColor3f (Color [0], Color [1], Color [2]);
   glVertex2f (0.25, 0.25);
glEnd;
```

Primitives



- Points

  GL_POINTS

- Lines

  GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP

- Triangles

  GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN

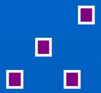
- Quadrilaterals and all other polygons

  GL_QUADS, GL_QUAD_STRIP, GL_POLYGON

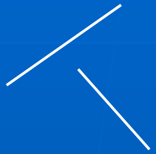
- Ordering of vertices (corners) defines front & back

  - GL_CCW <- Default
  - GL_CW

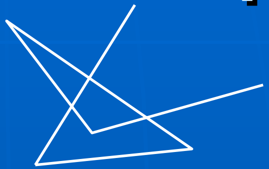
Primitives in OpenGL



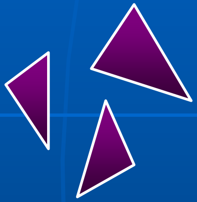
GL_POINTS



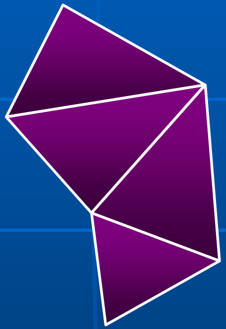
GL_LINES



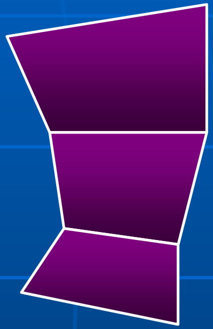
GL_LINE_STRIP



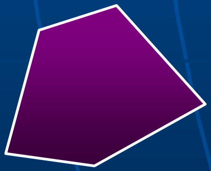
GL_TRIANGLES



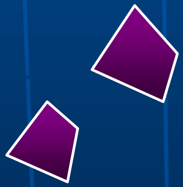
GL_TRIANGLE_STRIP



GL_QUAD_STRIPE

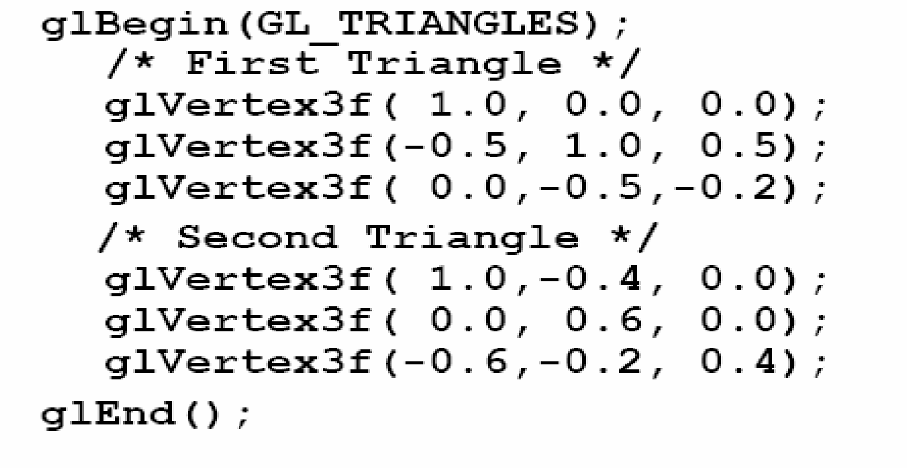


GL_POLIGON



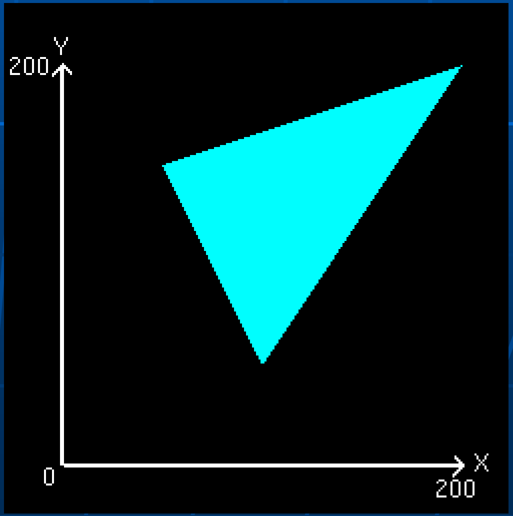
GL_QUADS

Triangles

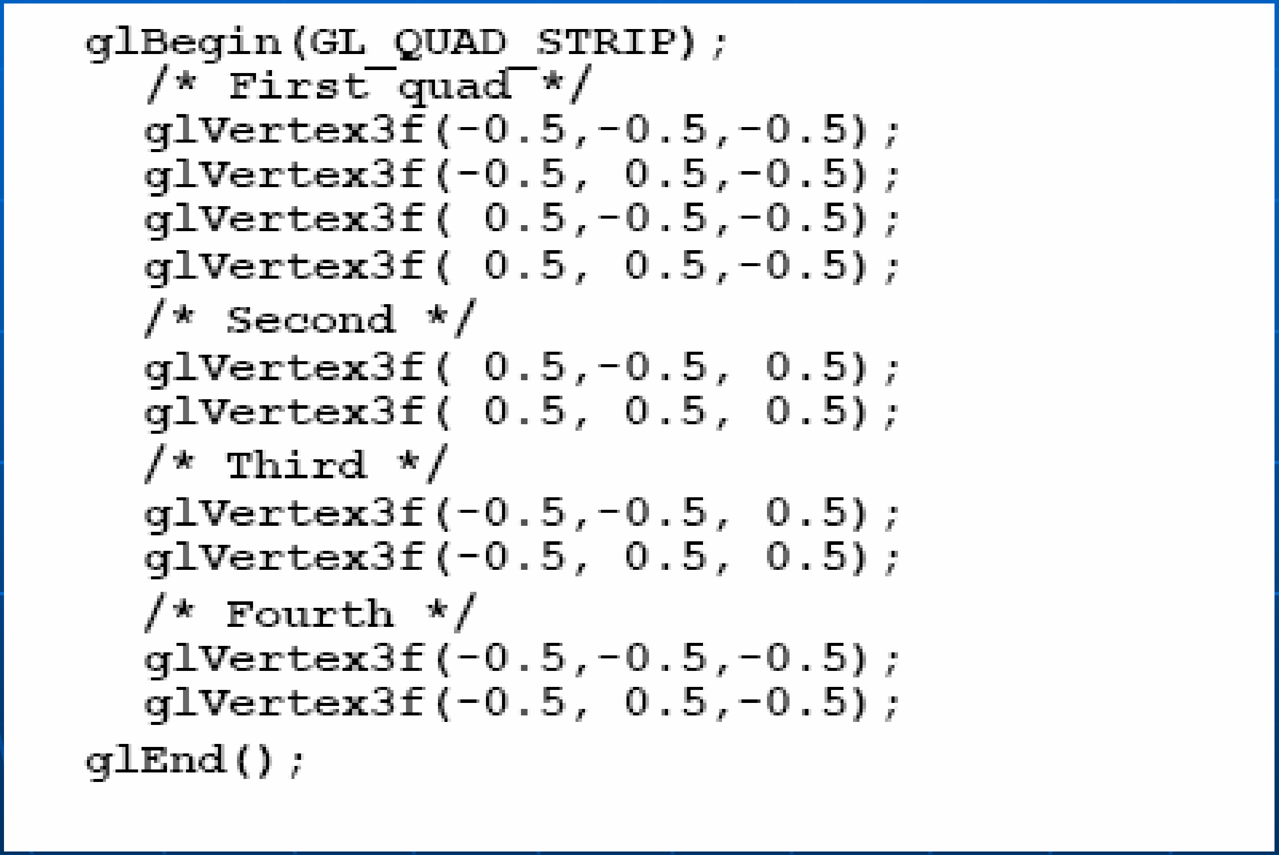


```
glBegin(GL_TRIANGLES);
   /* First Triangle */
   glVertex3f( 1.0, 0.0, 0.0);
   glVertex3f(-0.5, 1.0, 0.5);
   glVertex3f( 0.0,-0.5,-0.2);
  /* Second Triangle */
   glVertex3f( 1.0,-0.4, 0.0);
   glVertex3f( 0.0, 0.6, 0.0);
   glVertex3f(-0.6,-0.2, 0.4);
glEnd();
```

```
glBegin(GL_TRIANGLES);
      glColor2f(0.0f,1.0f);
      glVertex2f(150.0f, 50 .0f);
      glVertex2f(50.0f, 150 .0f);
      glVertex2f(200 .0f, 200 .0f);
glEnd();
```

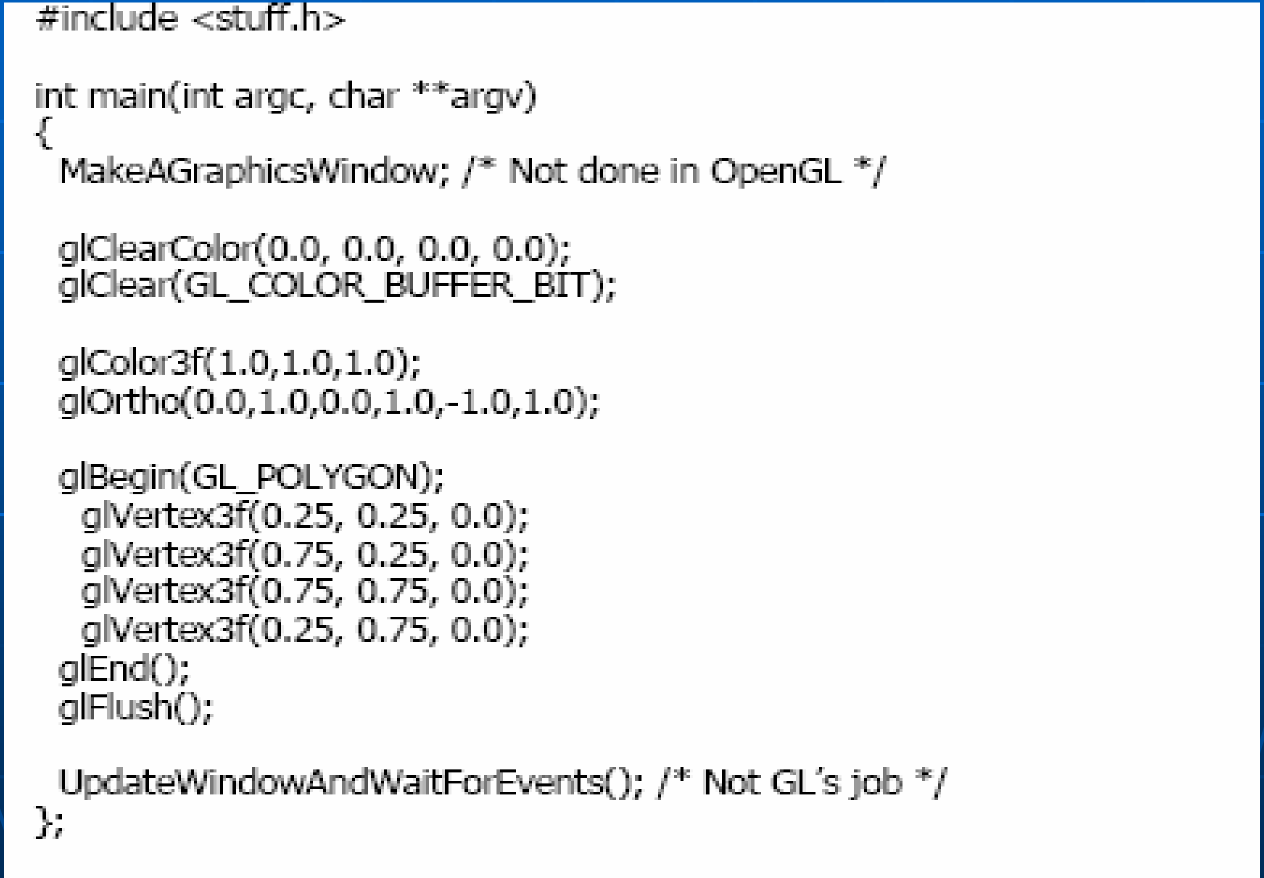


GL_QUAD_STRIPE



```
glBegin(GL_QUAD_STRIP);
  /* First quad */
  glVertex3f(-0.5,-0.5,-0.5);
  glVertex3f(-0.5, 0.5,-0.5);
  glVertex3f( 0.5,-0.5,-0.5);
  glVertex3f( 0.5, 0.5,-0.5);
  /* Second */
  glVertex3f( 0.5,-0.5, 0.5);
  glVertex3f( 0.5, 0.5, 0.5);
  /* Third */
  glVertex3f(-0.5,-0.5, 0.5);
  glVertex3f(-0.5, 0.5, 0.5);
  /* Fourth */
  glVertex3f(-0.5,-0.5,-0.5);
  glVertex3f(-0.5, 0.5,-0.5);
glEnd();
```

Sample pseudoprogram

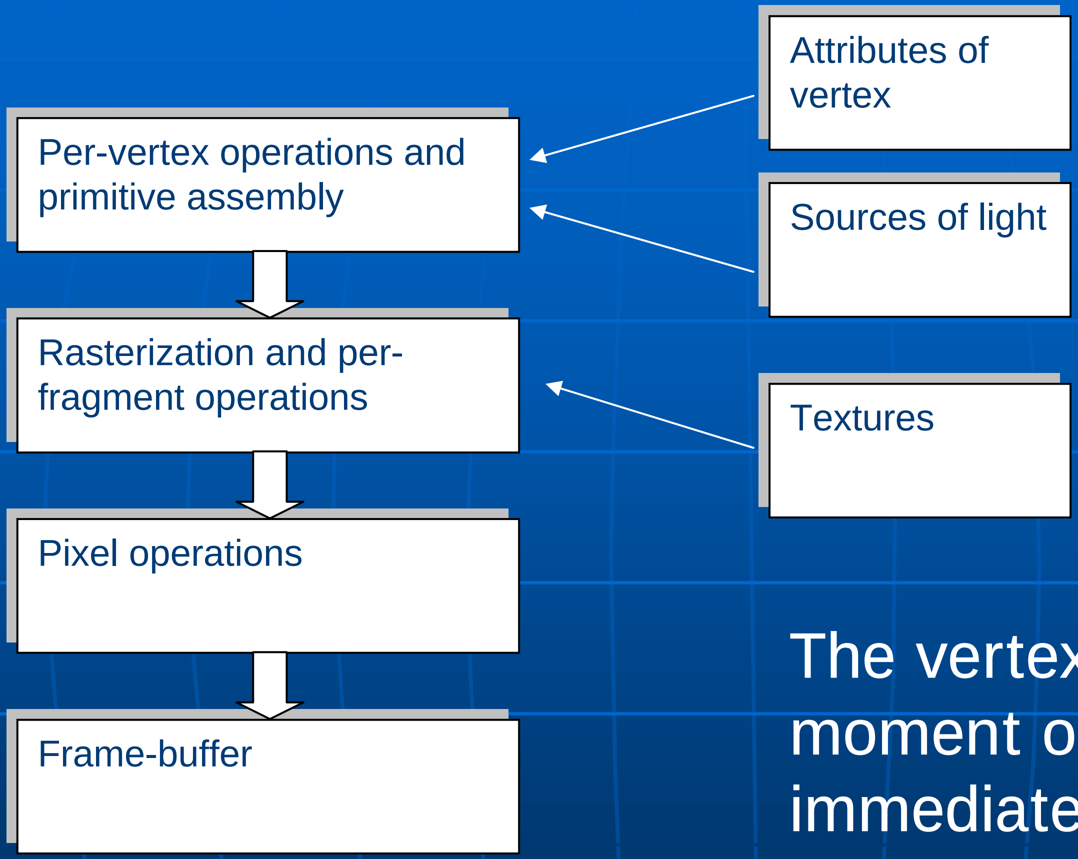


```
#include <stuff.h>

int main(int argc, char **argv)
{
  MakeAGraphicsWindow; /* Not done in OpenGL */

  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);

  glColor3f(1.0,1.0,1.0);
  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();

  UpdateWindowAndWaitForEvents(); /* Not GL's job */
};
```

Matrices of transformations

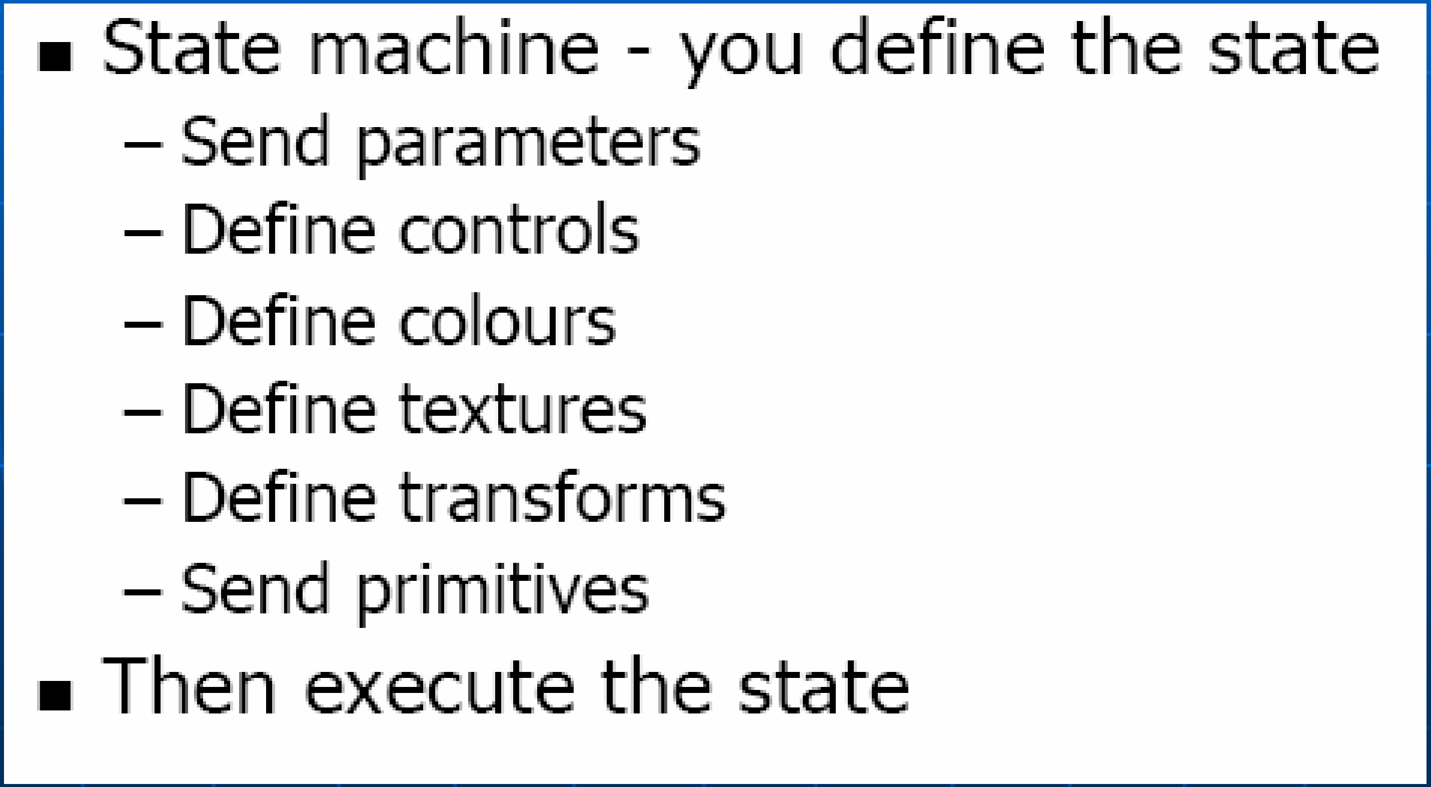
```
void glTranslated(GLdouble x,
                  GLdouble y,
                  GLdouble z);

void glScaled(GLdouble x,
              GLdouble y,
              GLdouble z);

void glRotated(GLdouble angle,
               GLdouble ax,
               GLdouble ay,
               GLdouble az);

void gluPerspective(GLdouble fov,
                    GLdouble aspect,
                    GLdouble znear,
                    GLdouble zfar);
```

Pipeline


Attributes of vertex
Sources of light
Textures
Per-vertex operations and primitive assembly
Rasterization and per-fragment operations
Pixel operations
Frame-buffer

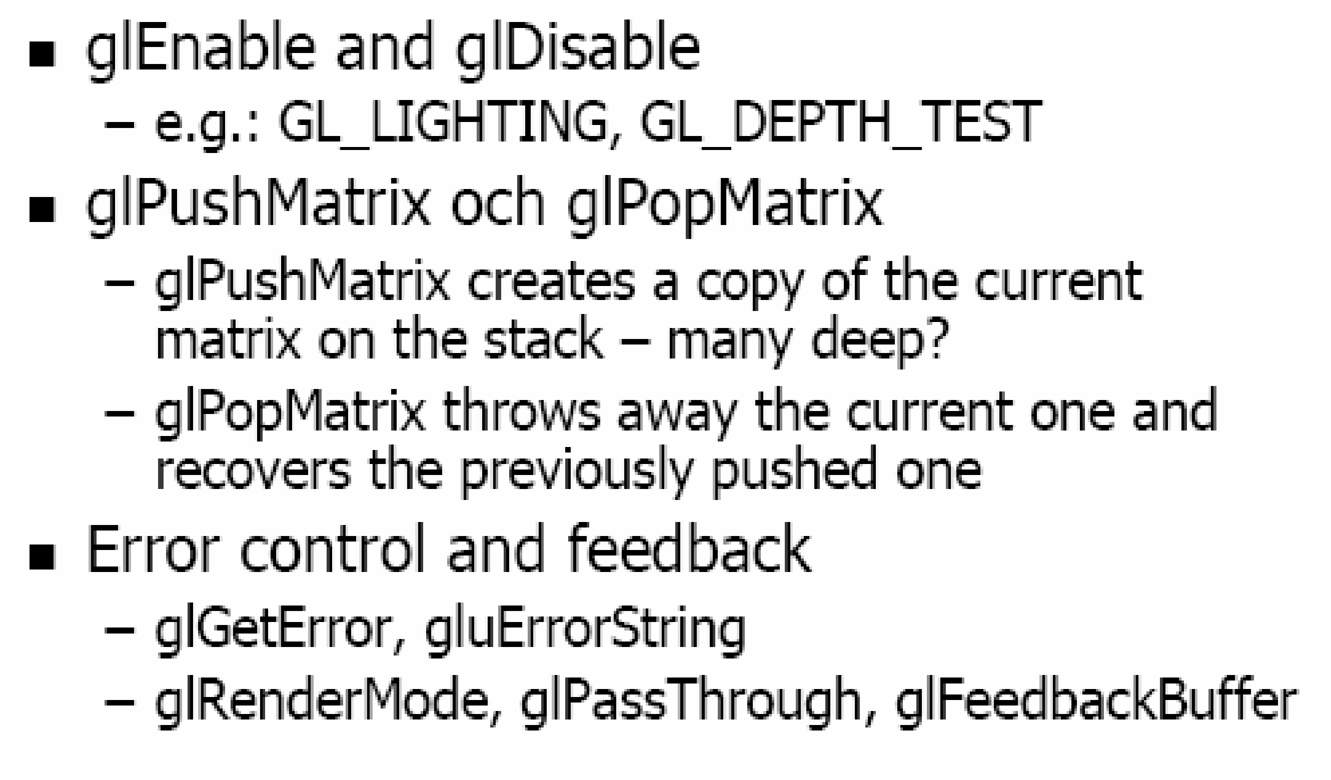
The vertex of any object at moment of definition immediately is putting to pipeline and is passed all stages of processing

The GL state machine



- State machine - you define the state
  - Send parameters
  - Define controls
  - Define colours
  - Define textures
  - Define transforms
  - Send primitives
- Then execute the state

Controlling the state



- glEnable and glDisable
  - e.g.: GL_LIGHTING, GL_DEPTH_TEST
- glPushMatrix och glPopMatrix
  - glPushMatrix creates a copy of the current matrix on the stack – many deep?
  - glPopMatrix throws away the current one and recovers the previously pushed one
- Error control and feedback
  - glGetError, gluErrorString
  - glRenderMode, glPassThrough, glFeedbackBuffer

Attributes of vertex

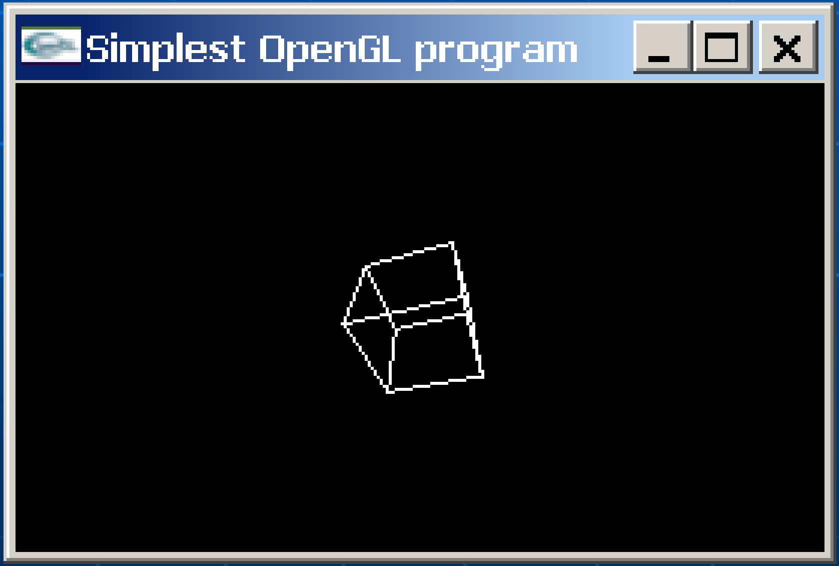
- Every vertex besides *position* may have some other attributes:
  - Material
  - Color
  - Normal
  - Texture coordinates
- Always current values of attributes are used

Process of visualization

- Define window for drawing
- Define constant attributes and properties (sources of lights, textures and so on)
- For every frame
  - Clear frame-buffer
  - Define position of camera
  - For every object
    - Define transformation
    - Get attributes
    - Get geometry
  - Update window

Define window

- glViewport(x, y, width, height)



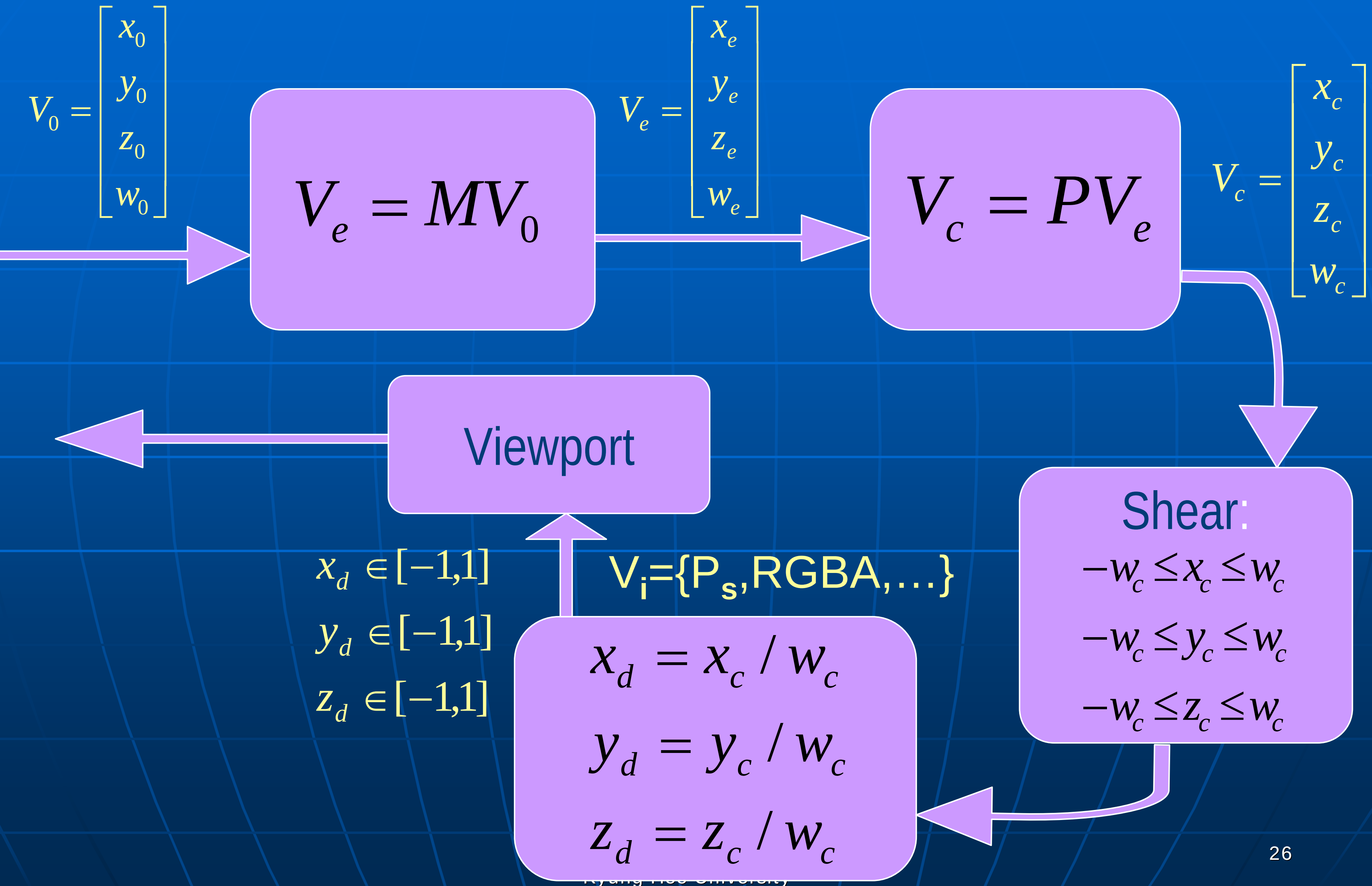
Operations with frame-buffer

- definition of color for filling of frame-buffer
  - glClearColor(red, green, blue, alpha)

    red,green,blue,alpha $\in [0,1]$

- Fill screen buffers
  - glClear(GL_COLOR_BUFFER_BIT);

Transformation of coordinates



$$V_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}$$

$$V_e = MV_0$$

$$V_e = \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}$$

$$V_c = PV_e$$

$$V_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix}$$

Shear:

$$-w_c \le x_c \le w_c$$

$$-w_c \le y_c \le w_c$$

$$-w_c \le z_c \le w_c$$

$$x_d = x_c / w_c$$

$$y_d = y_c / w_c$$

$$z_d = z_c / w_c$$

V_i={P_s,RGBA,…}

$$x_d \in [-1,1]$$

$$y_d \in [-1,1]$$

$$z_d \in [-1,1]$$

Viewport

Matrices of transformations

- select the matrix for transformation:

```
void glMatrixMode(Glenum mode);
    mode={GL_MODELVIEW|GL_PROJECTION}
```

- Two main operations on matrices:

```
void glLoadIdentity();
```

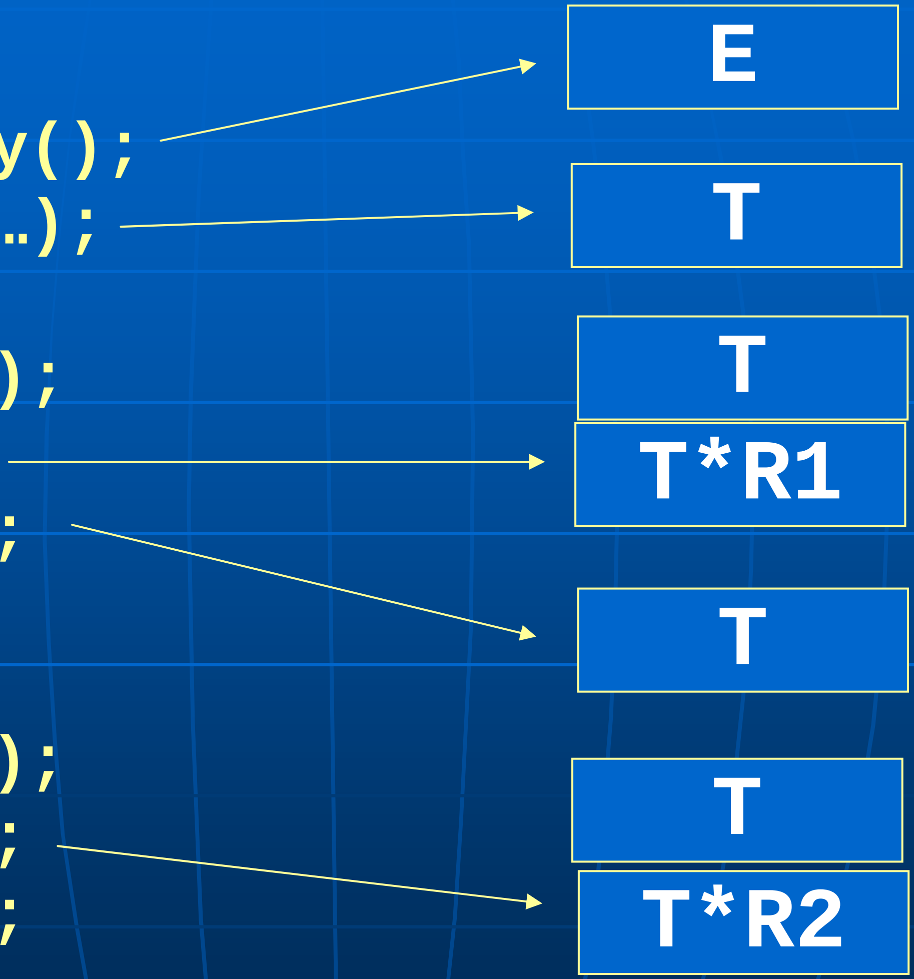
$$M = E$$

```
void glMultMatrixd(GLdouble c[16]);
```

$$M = M \cdot \begin{bmatrix} c[0] & c[4] & c[8] & c[12] \\ c[1] & c[5] & c[9] & c[13] \\ c[2] & c[6] & c[10] & c[14] \\ c[3] & c[7] & c[11] & c[15] \end{bmatrix}$$

Stack of matrices

```
glLoadIdentity();
glTranslated(…);

glPushMatrix();
glRotated(…);
glPopMatrix();

glPushMatrix();
glRotataed(…);
glPopMatrix();
```



E

T

T

T*R1

T

T

T*R2

Transformations

$$M = M_{view} \cdot M_{mdl}$$

M_{mdl} - glTranslate, glRotate, glScale

M_{view} - gluLookAt(eye$_x$, eye$_y$, eye$_z$,
aim$_x$, aim$_y$, aim$_z$,
up$_x$, up$_y$, up$_z$)

Eye – coordinates of camera
Aim – coordinates of target
Up – direction to up

Transformations (2)

- glMatrixMode(GL_MODELVIEW);
- gluLookAt(..);

Camera

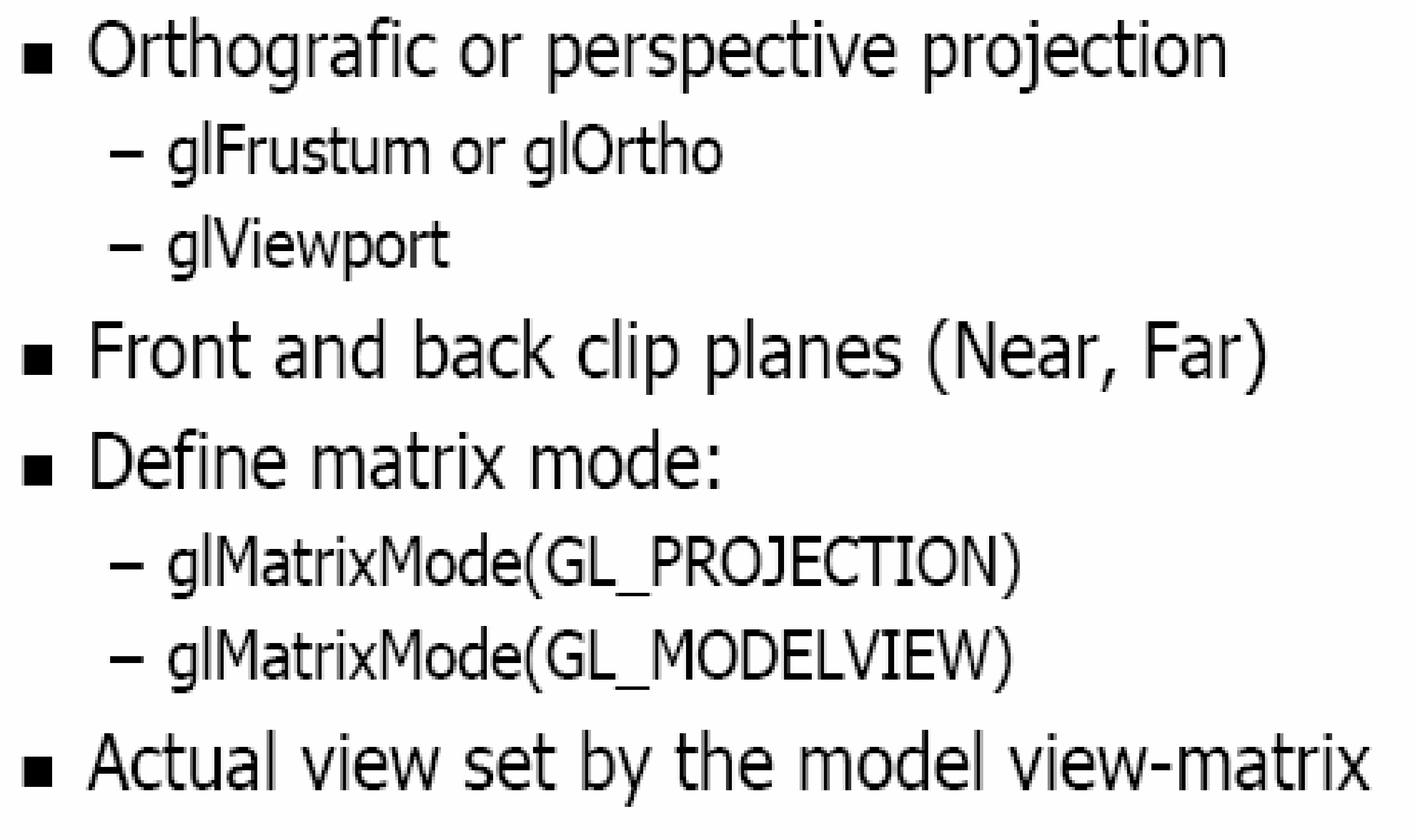
- glTranslate(...);
- glRotate(...);
- glTranslate(...);

Model transformation

- glBegin(...);
- ...
- glEnd();

Geometry

Camera and projection matrix



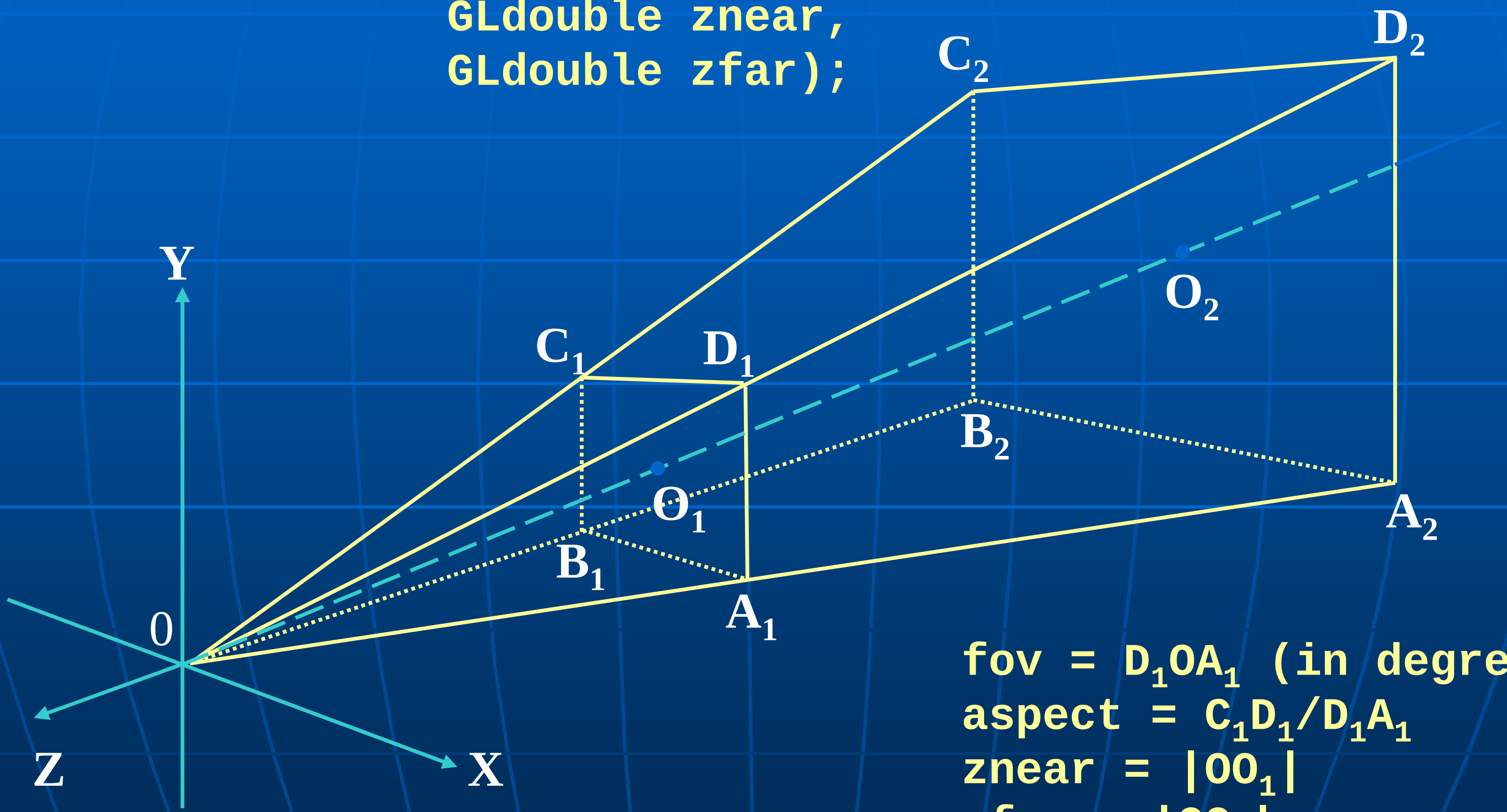
- Orthografic or perspective projection
  - glFrustum or glOrtho
  - glViewport
- Front and back clip planes (Near, Far)
- Define matrix mode:
  - glMatrixMode(GL_PROJECTION)
  - glMatrixMode(GL_MODELVIEW)
- Actual view set by the model view-matrix

Perspective projection

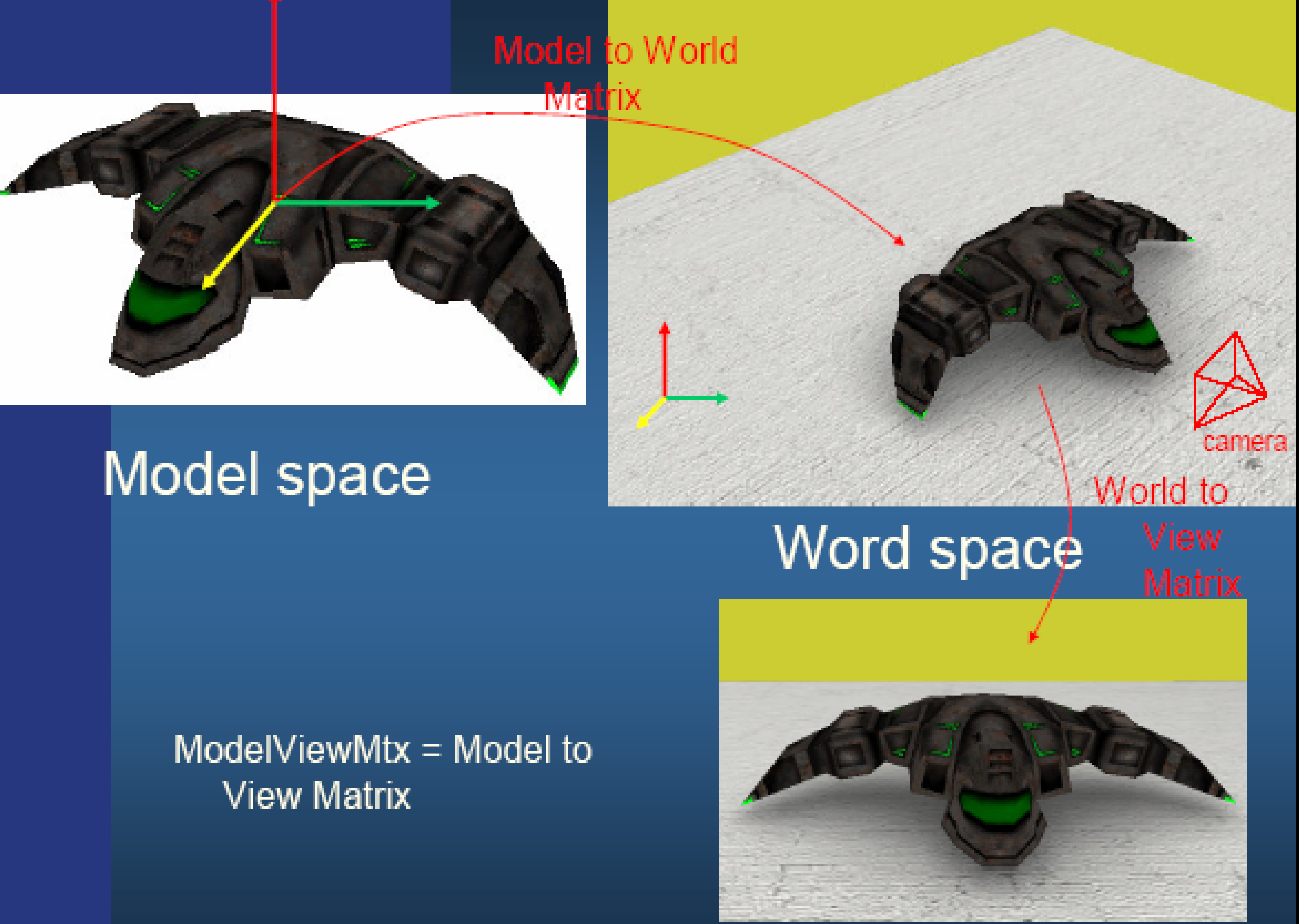
- glMatrixModel(GL_PROJECTION);
- gluPerspective(...)

gluPerspective

```
void gluPerspective(GLdouble fov,
                    GLdouble aspect,
                    GLdouble znear,
                    GLdouble zfar);
```



fov = D_1OA_1 (in degrees)
aspect = C_1D_1/D_1A_1
znear = $|OO_1|$
zfar = $|OO_2|$



Model to World Matrix

Model space

Word space

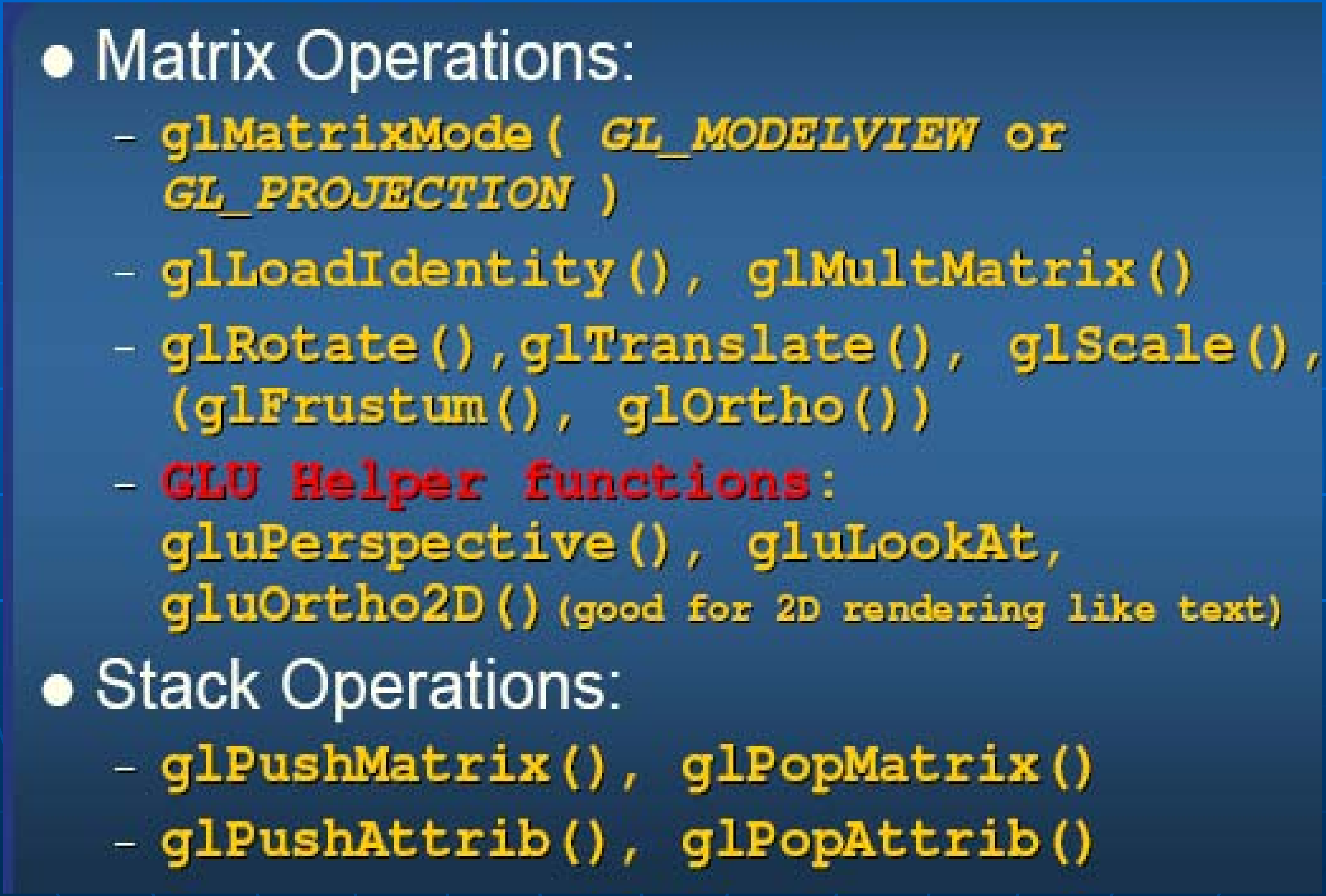
camera

World to View Matrix

View space

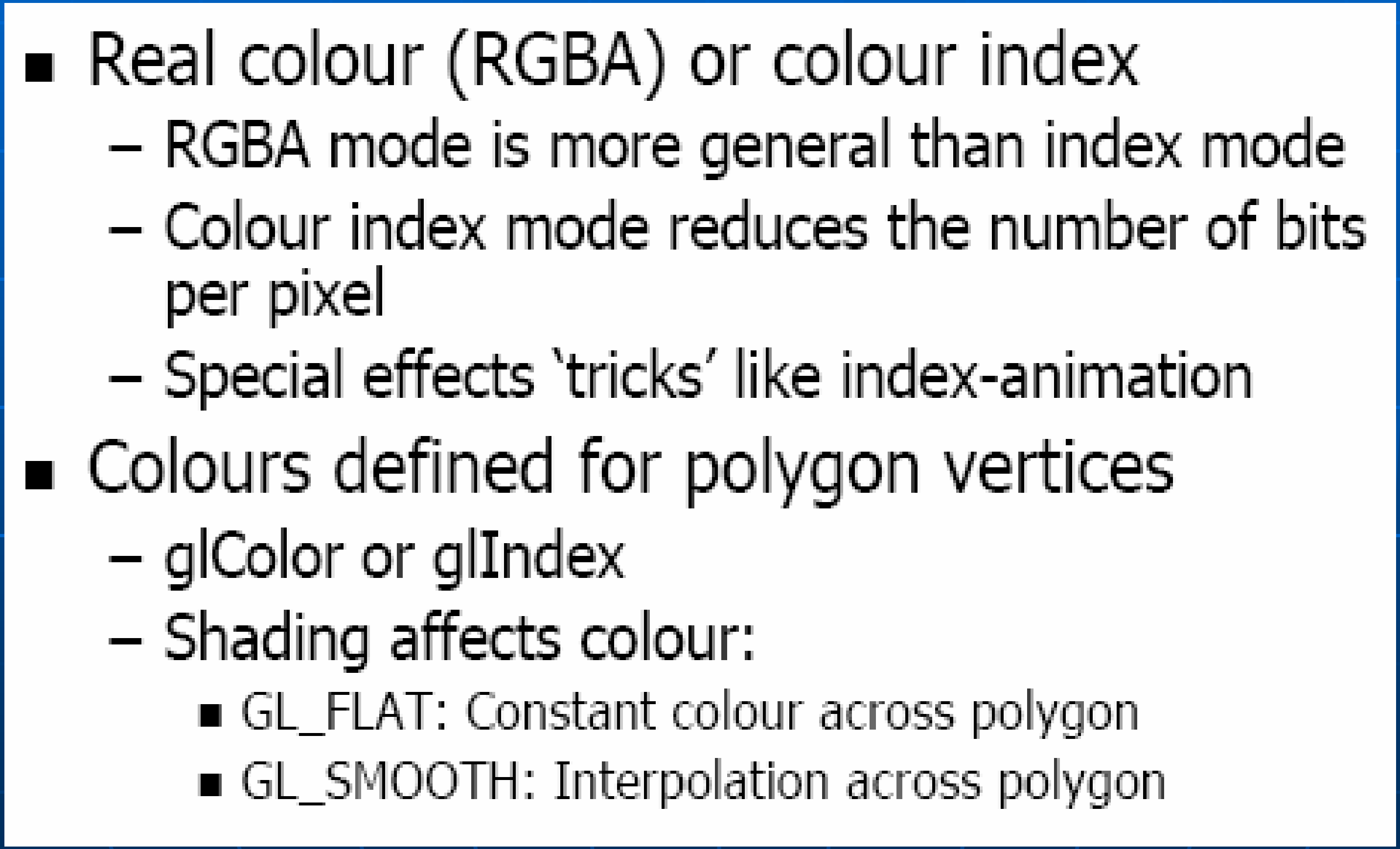
ModelViewMtx = Model to View Matrix

Demo



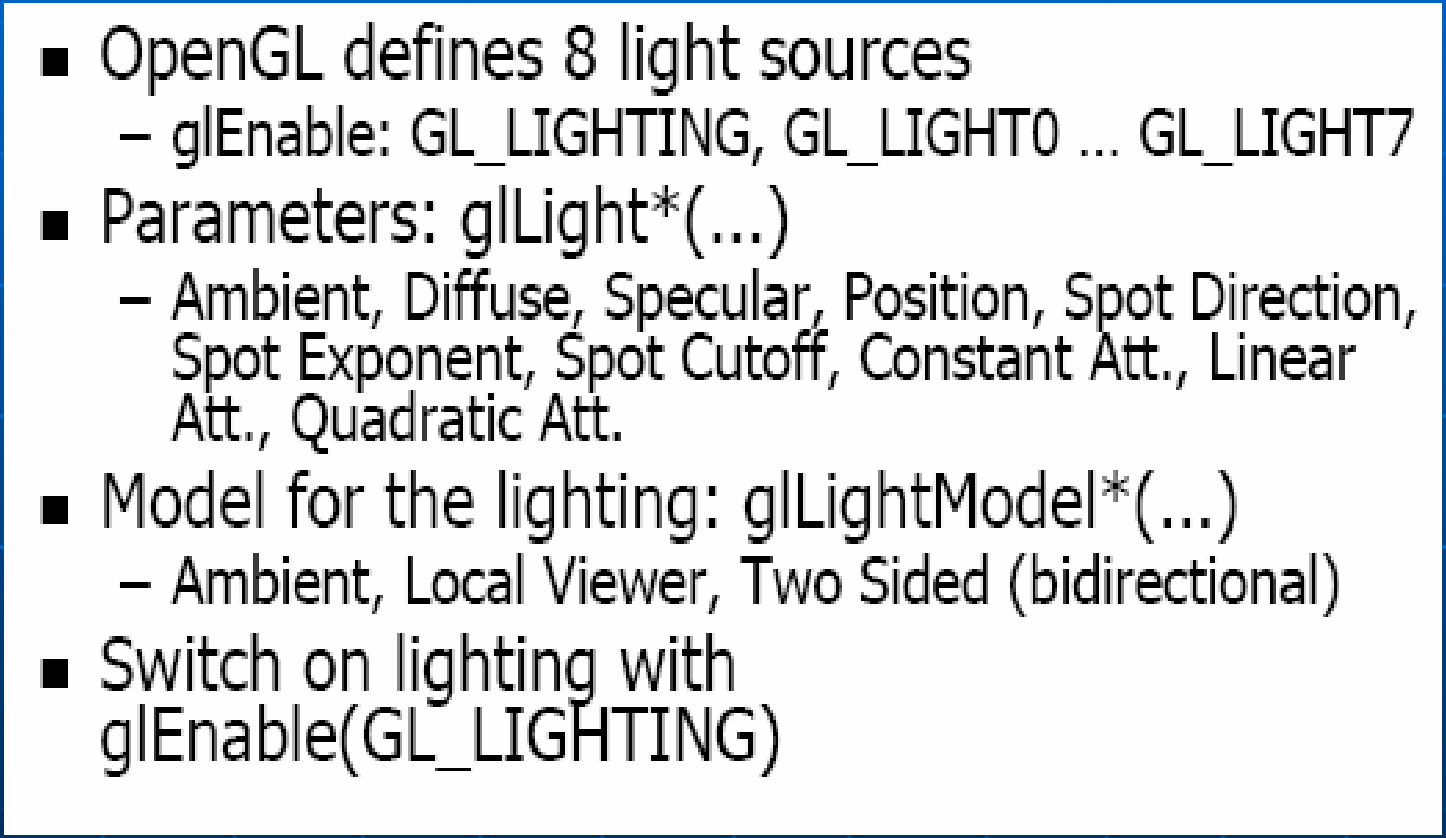
- Matrix Operations:
  - glMatrixMode(*GL_MODELVIEW* or *GL_PROJECTION*)
  - glLoadIdentity(), glMultMatrix()
  - glRotate(),glTranslate(), glScale(), (glFrustum(), glOrtho())
  - GLU Helper functions: gluPerspective(), gluLookAt, gluOrtho2D() (good for 2D rendering like text)
- Stack Operations:
  - glPushMatrix(), glPopMatrix()
  - glPushAttrib(), glPopAttrib()

Color

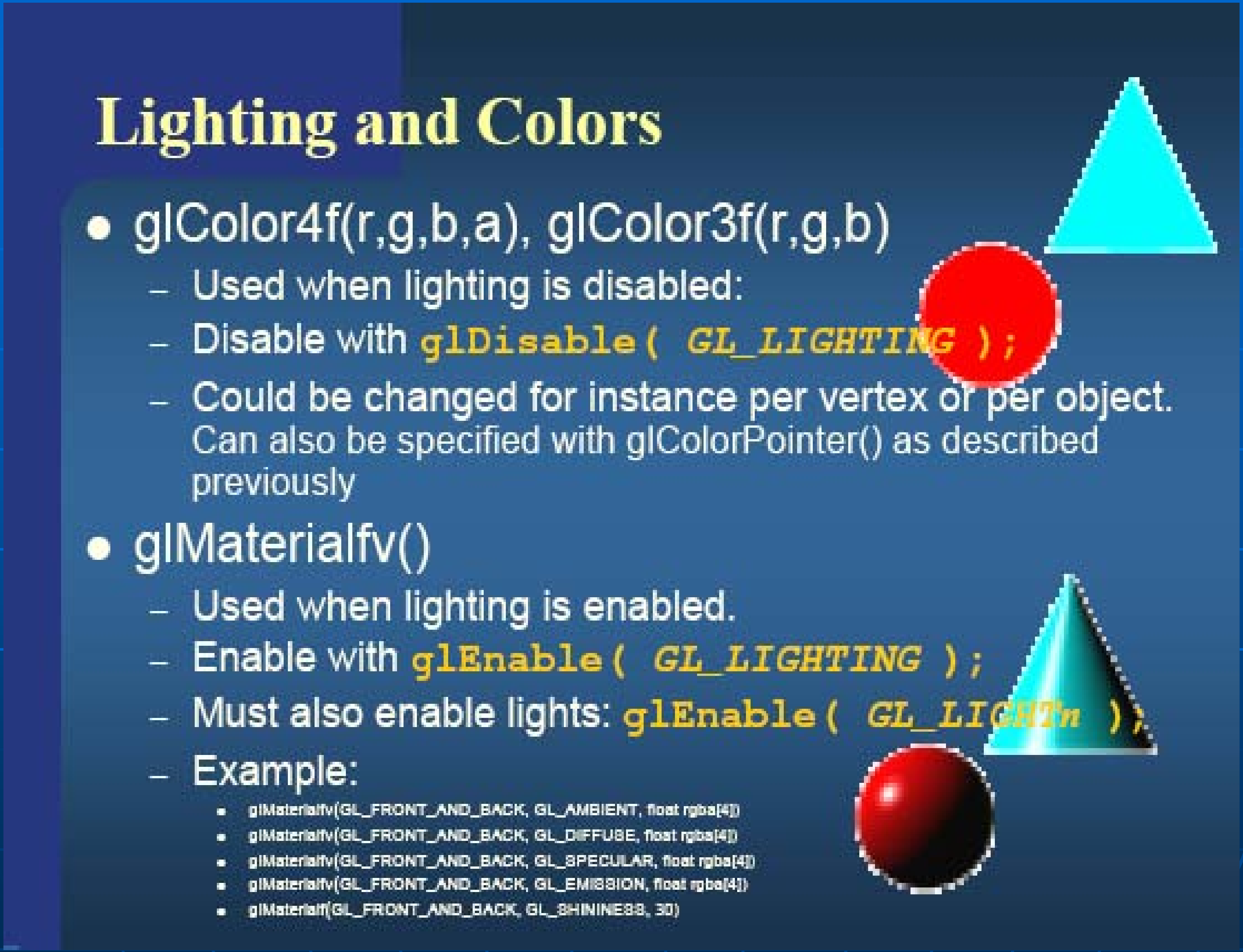


- Real colour (RGBA) or colour index
  - – RGBA mode is more general than index mode
  - – Colour index mode reduces the number of bits per pixel
  - – Special effects 'tricks' like index-animation
- Colours defined for polygon vertices
  - – glColor or glIndex
  - – Shading affects colour:
    - GL_FLAT: Constant colour across polygon
    - GL_SMOOTH: Interpolation across polygon

Light



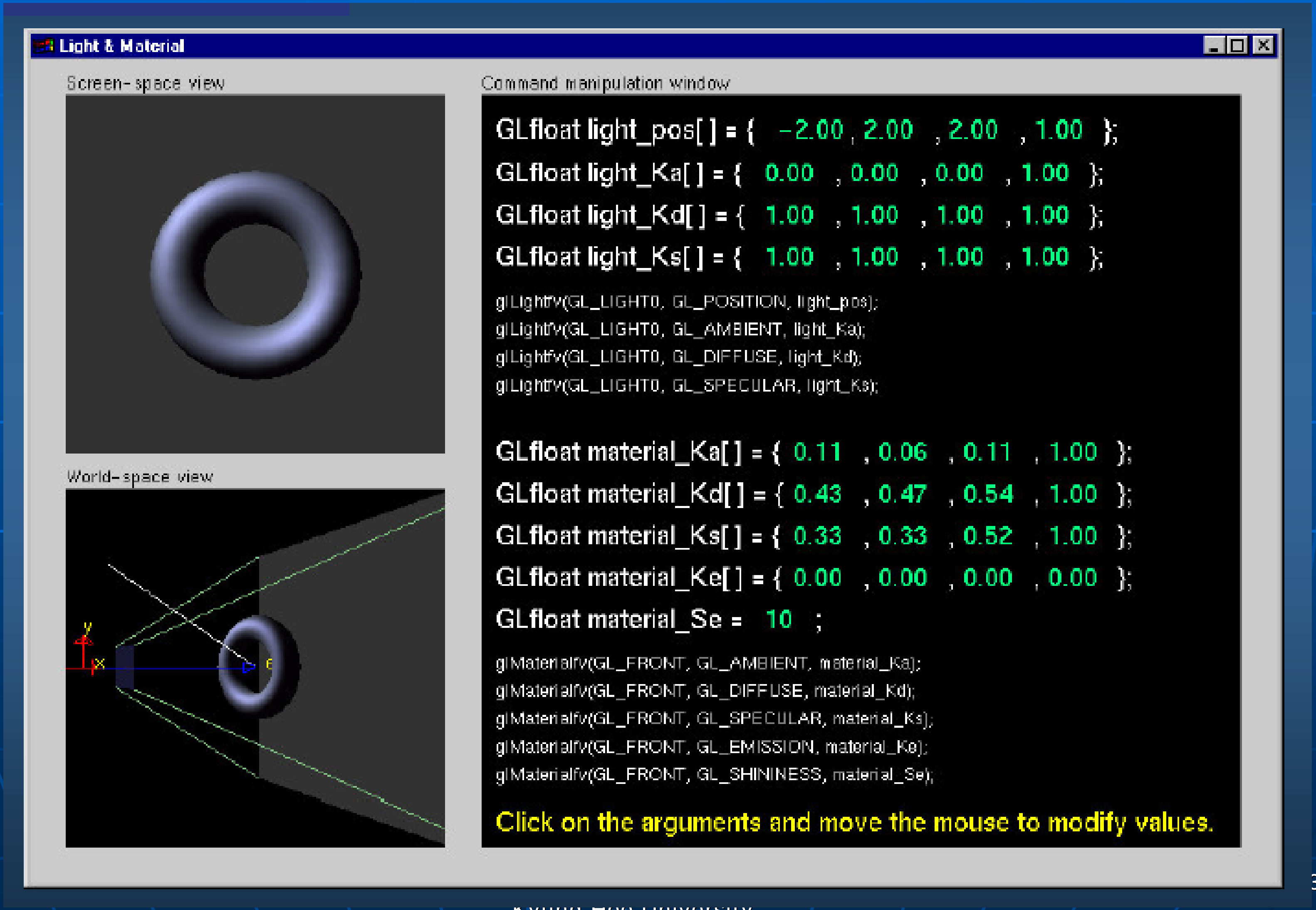
- OpenGL defines 8 light sources
  - glEnable: GL_LIGHTING, GL_LIGHT0 ... GL_LIGHT7
- Parameters: glLight*(...)
  - Ambient, Diffuse, Specular, Position, Spot Direction, Spot Exponent, Spot Cutoff, Constant Att., Linear Att., Quadratic Att.
- Model for the lighting: glLightModel*(...)
  - Ambient, Local Viewer, Two Sided (bidirectional)
- Switch on lighting with glEnable(GL_LIGHTING)



Lighting and Colors

- glColor4f(r,g,b,a), glColor3f(r,g,b)
  - Used when lighting is disabled:
  - Disable with `glDisable( GL_LIGHTING );`
  - Could be changed for instance per vertex or per object. Can also be specified with glColorPointer() as described previously
- glMaterialfv()
  - Used when lighting is enabled.
  - Enable with `glEnable( GL_LIGHTING );`
  - Must also enable lights: `glEnable( GL_LIGHTn );`
  - Example:
    - glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, float rgba[4])
    - glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, float rgba[4])
    - glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, float rgba[4])
    - glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, float rgba[4])
    - glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 30)

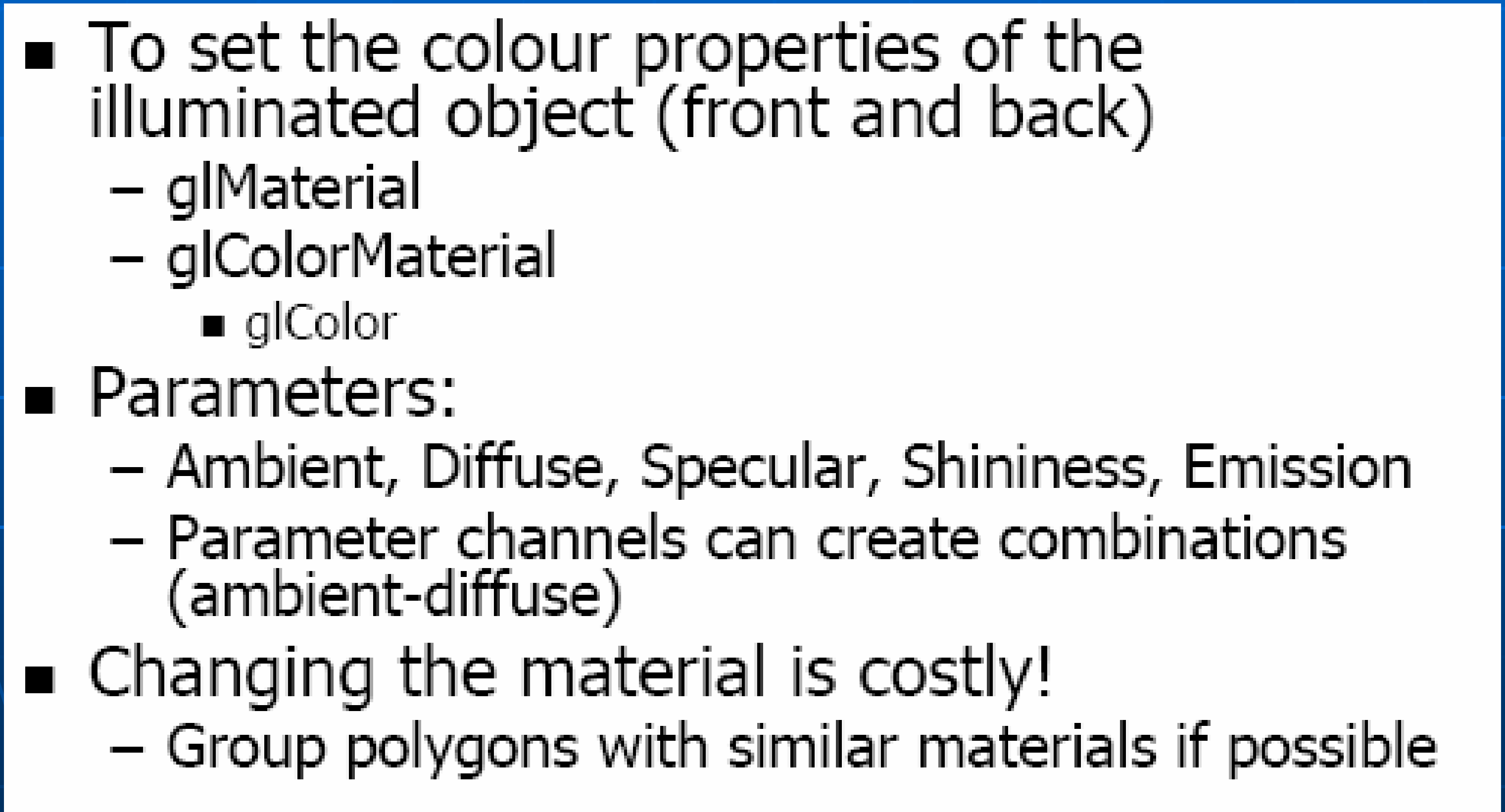
Example of lighting



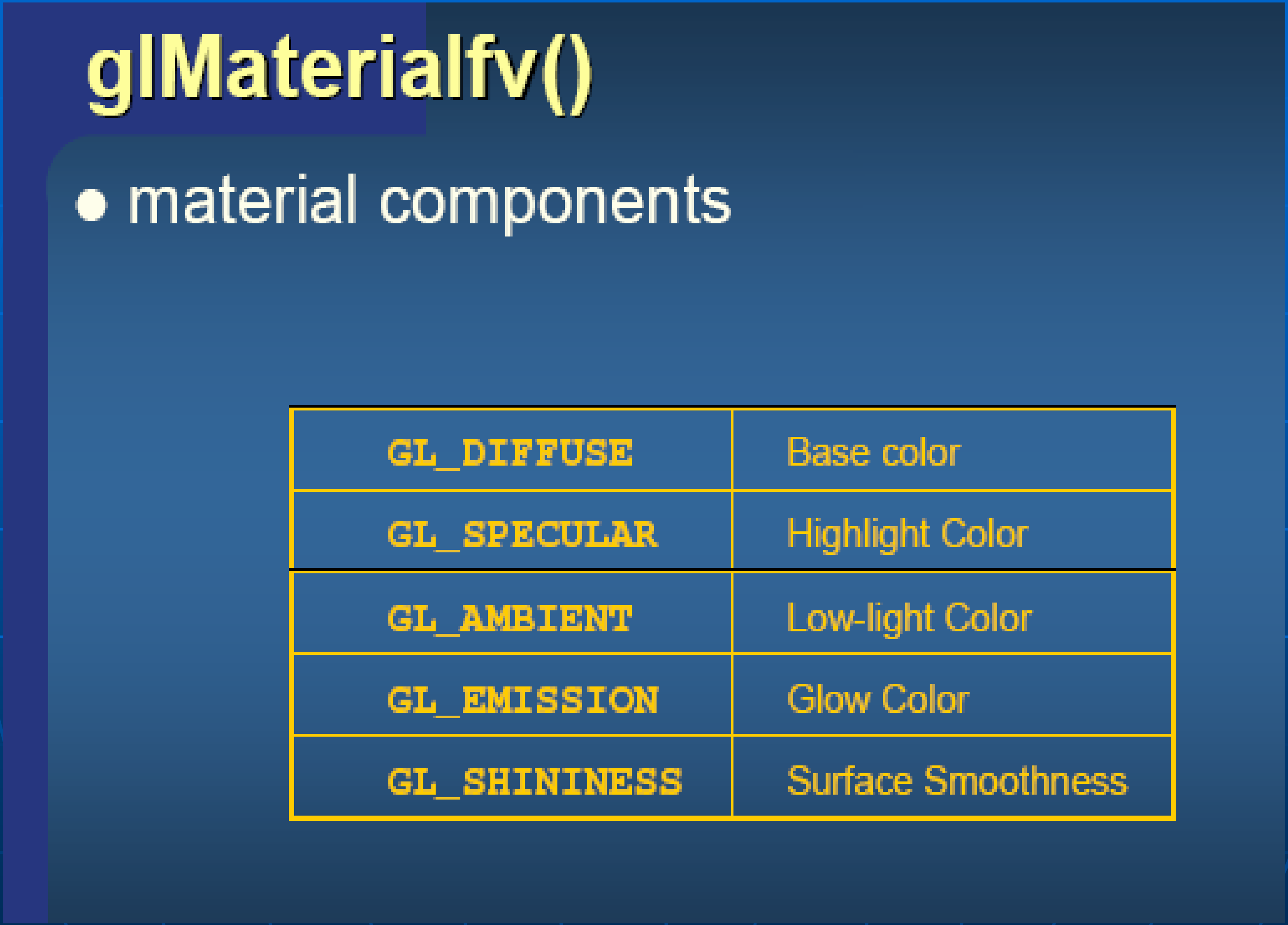
```
GLfloat light_pos[ ] = {  -2.00 , 2.00  , 2.00  , 1.00  };
GLfloat light_Ka[ ] = {  0.00  , 0.00  , 0.00  , 1.00  };
GLfloat light_Kd[ ] = {  1.00  , 1.00  , 1.00  , 1.00  };
GLfloat light_Ks[ ] = {  1.00  , 1.00  , 1.00  , 1.00  };
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[ ] = { 0.11  , 0.06  , 0.11  , 1.00  };
GLfloat material_Kd[ ] = { 0.43  , 0.47  , 0.54  , 1.00  };
GLfloat material_Ks[ ] = { 0.33  , 0.33  , 0.52  , 1.00  };
GLfloat material_Ke[ ] = { 0.00  , 0.00  , 0.00  , 0.00  };
GLfloat material_Se =  10  ;
glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
glMaterialfv(GL_FRONT, GL_SHININESS, material_Se);
```

Click on the arguments and move the mouse to modify values.

Material



- To set the colour properties of the illuminated object (front and back)
  - glMaterial
  - glColorMaterial
    - glColor
- Parameters:
  - Ambient, Diffuse, Specular, Shininess, Emission
  - Parameter channels can create combinations (ambient-diffuse)
- Changing the material is costly!
  - Group polygons with similar materials if possible

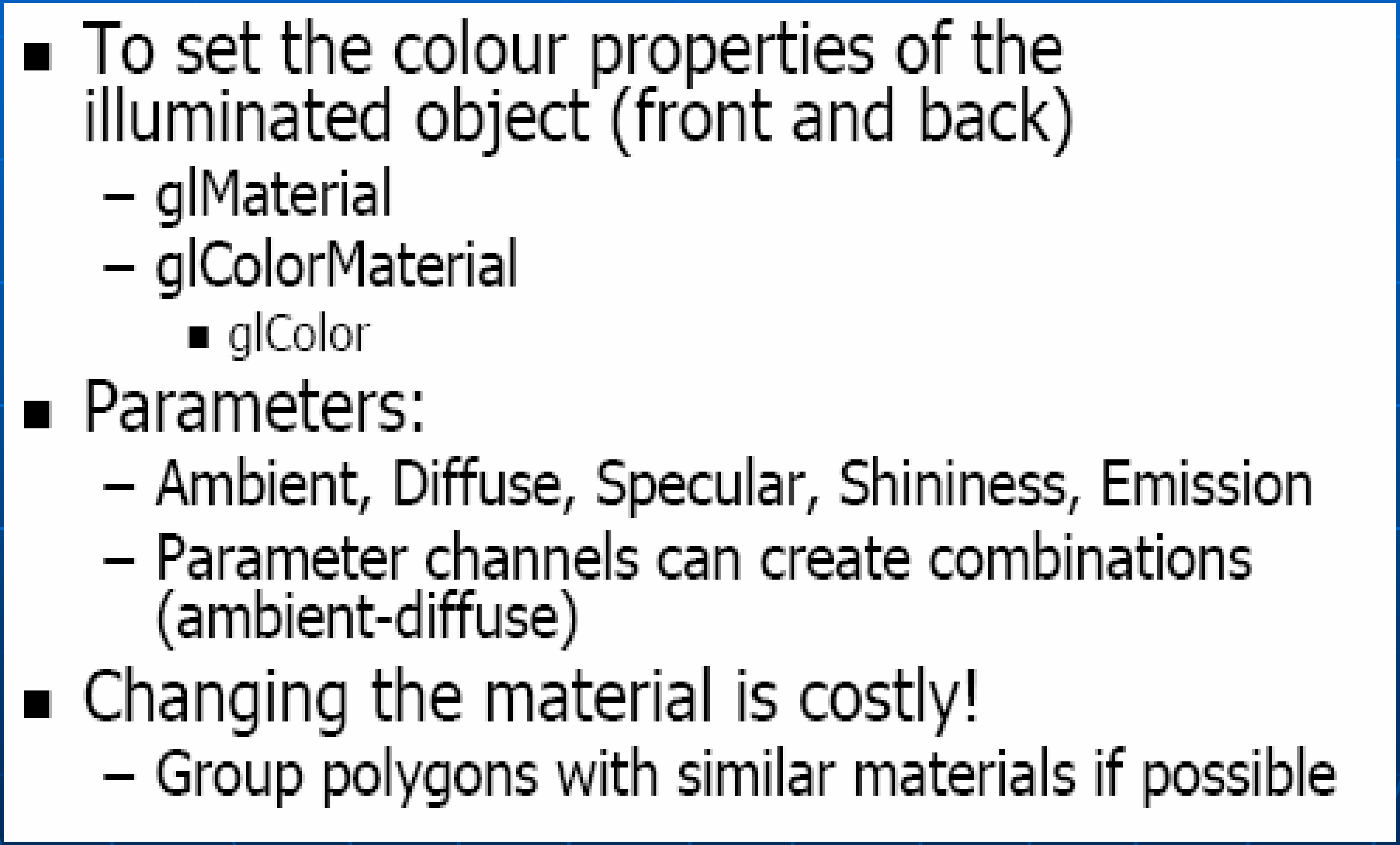


glMaterialfv()

- material components

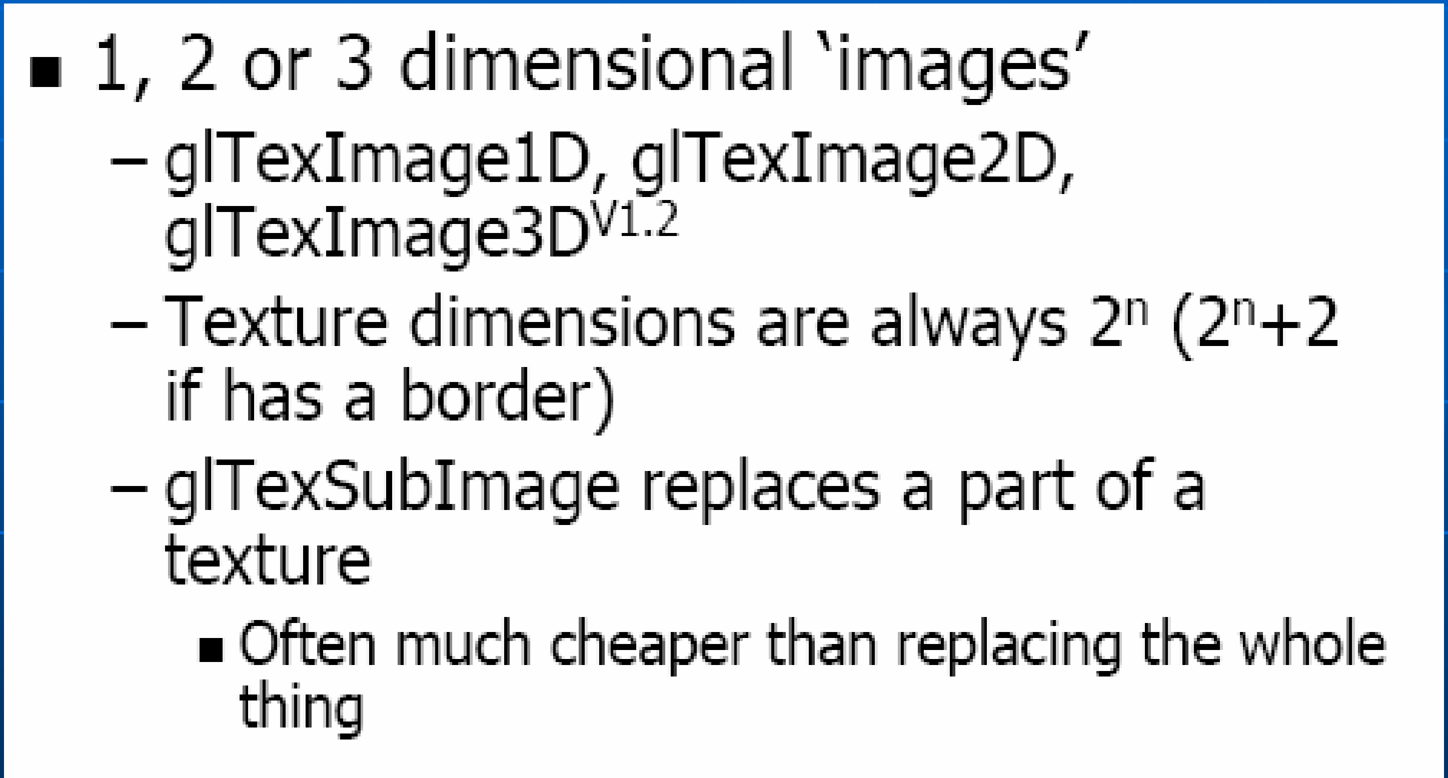
GL_DIFFUSE	Base color
GL_SPECULAR	Highlight Color
GL_AMBIENT	Low-light Color
GL_EMISSION	Glow Color
GL_SHININESS	Surface Smoothness

Shading



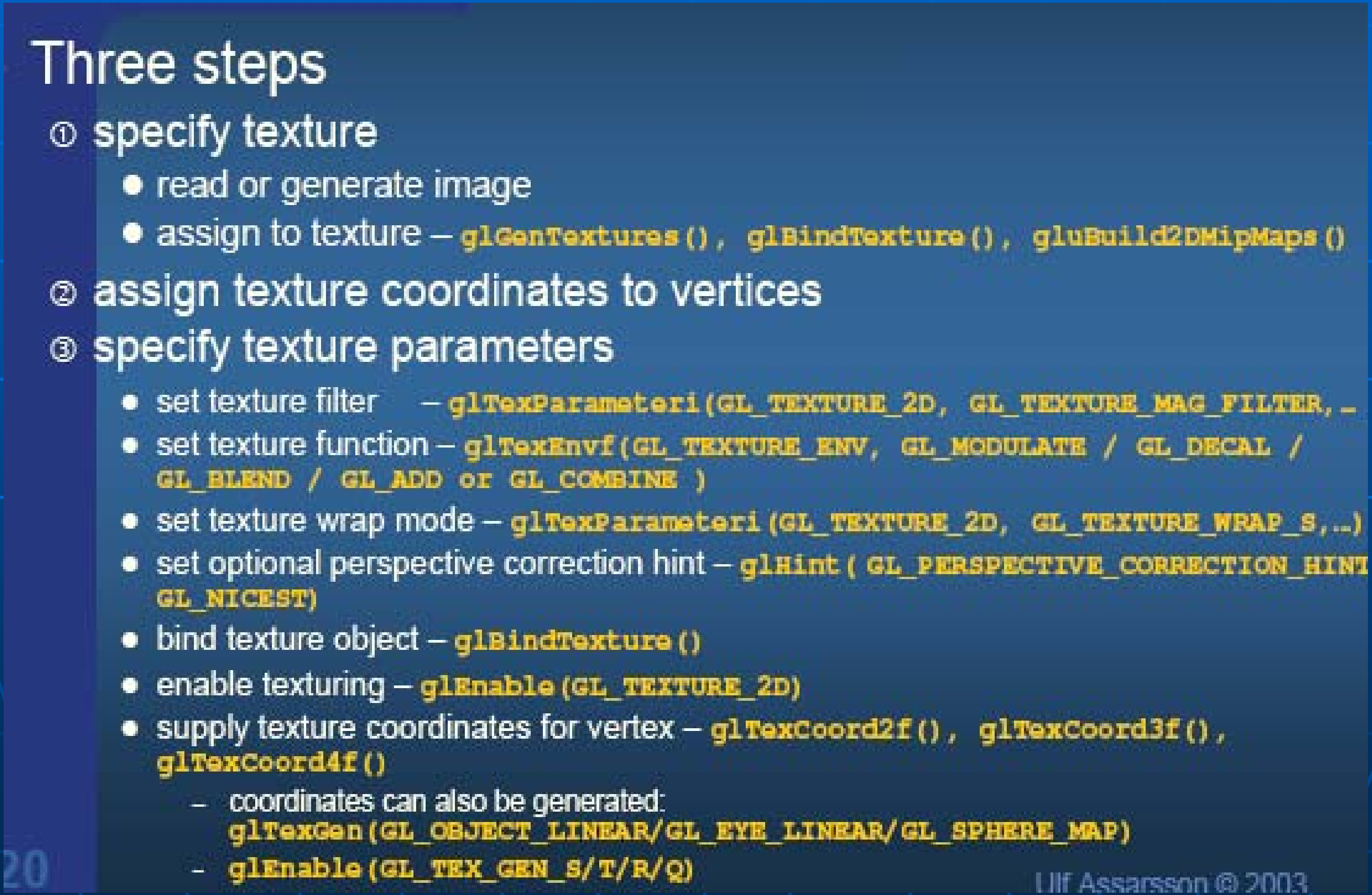
- To set the colour properties of the illuminated object (front and back)
  - glMaterial
  - glColorMaterial
    - glColor
- Parameters:
  - Ambient, Diffuse, Specular, Shininess, Emission
  - Parameter channels can create combinations (ambient-diffuse)
- Changing the material is costly!
  - Group polygons with similar materials if possible

Textures



- 1, 2 or 3 dimensional ‘images’
  - – glTexImage1D, glTexImage2D, glTexImage3D$^{V1.2}$
  - – Texture dimensions are always 2^n (2^n+2 if has a border)
  - – glTexSubImage replaces a part of a texture
    - Often much cheaper than replacing the whole thing

Textures (2)



Three steps

① specify texture

- read or generate image
- assign to texture – `glGenTextures(), glBindTexture(), gluBuild2DMipMaps()`

② assign texture coordinates to vertices

③ specify texture parameters

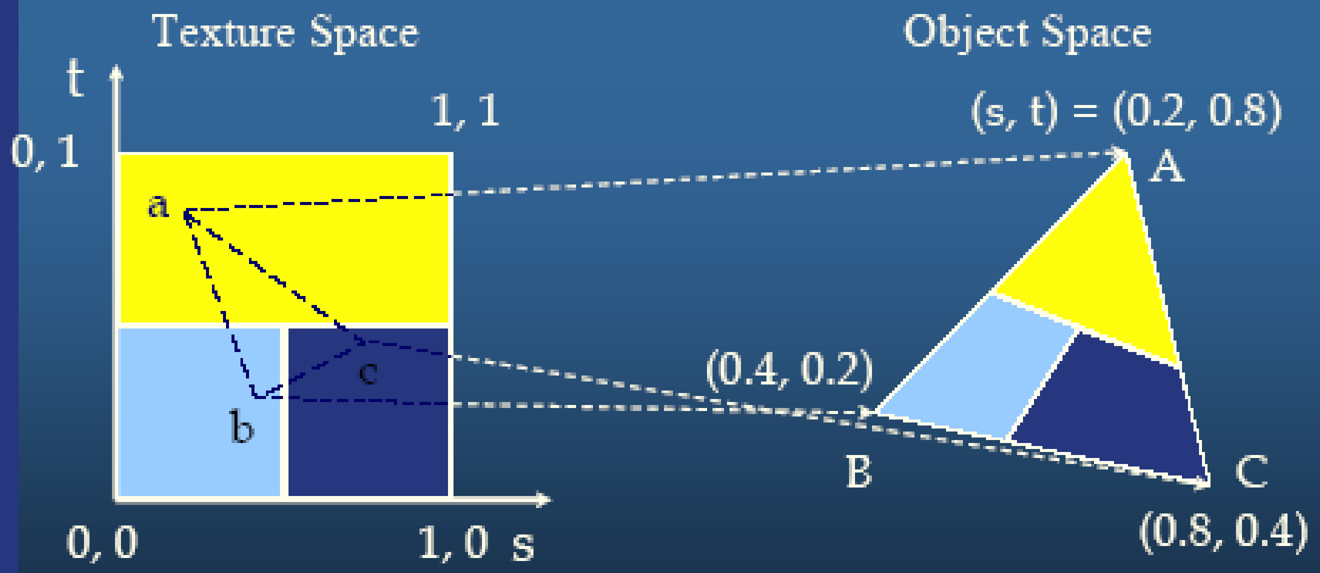
- set texture filter – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, …`
- set texture function – `glTexEnvf(GL_TEXTURE_ENV, GL_MODULATE / GL_DECAL / GL_BLEND / GL_ADD or GL_COMBINE )`
- set texture wrap mode – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,…)`
- set optional perspective correction hint – `glHint( GL_PERSPECTIVE_CORRECTION_HINT GL_NICEST)`
- bind texture object – `glBindTexture()`
- enable texturing – `glEnable(GL_TEXTURE_2D)`
- supply texture coordinates for vertex – `glTexCoord2f(), glTexCoord3f(), glTexCoord4f()`
    - coordinates can also be generated: `glTexGen(GL_OBJECT_LINEAR/GL_EYE_LINEAR/GL_SPHERE_MAP)`
    - `glEnable(GL_TEX_GEN_S/T/R/Q)`

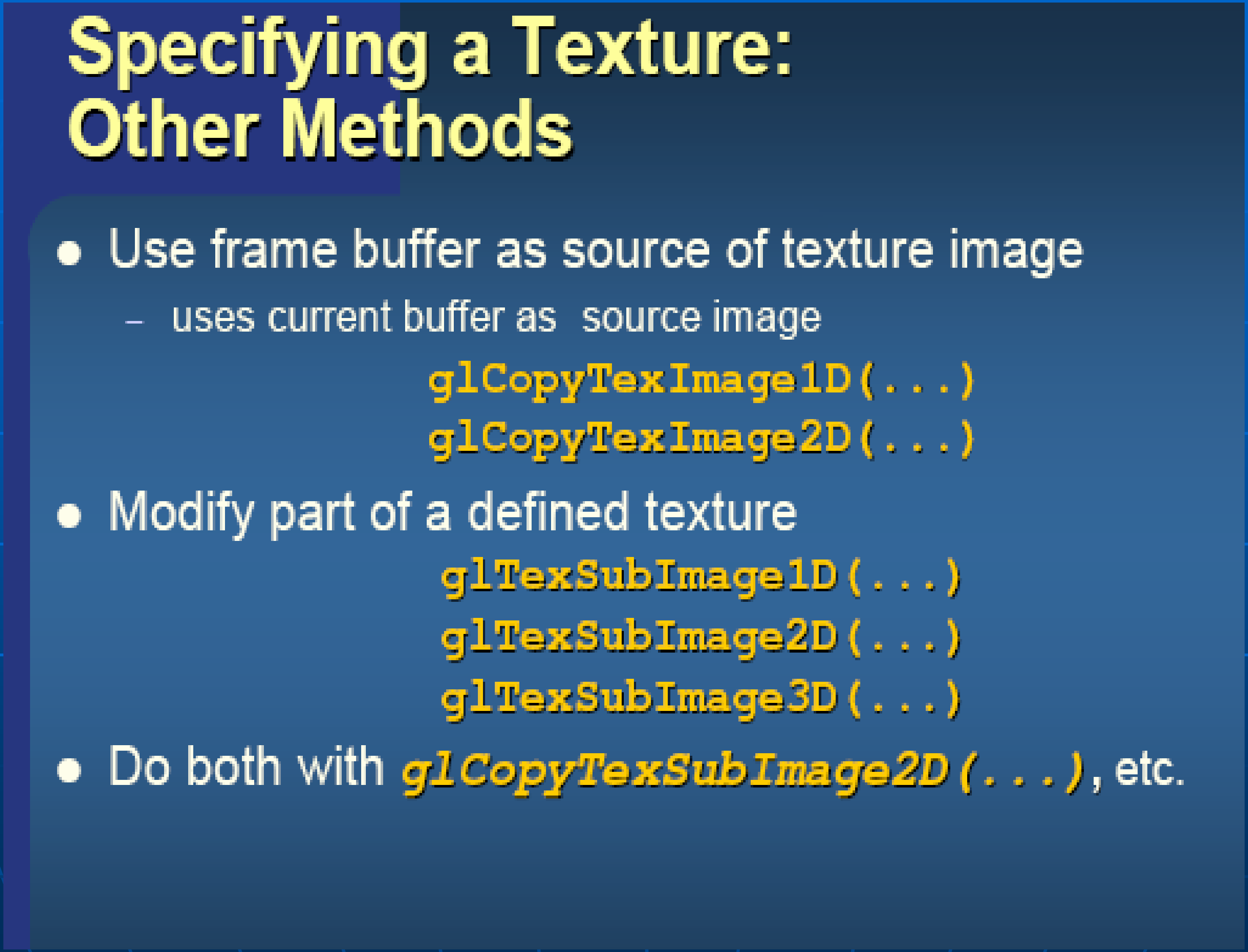
20

Ulf Assarsson © 2003

Assigning Texture coordinates - glTexCoord()

- Based on parametric texture coordinates
- `glTexCoord2f()` specified at each vertex





Specifying a Texture: Other Methods

- Use frame buffer as source of texture image
  - uses current buffer as source image

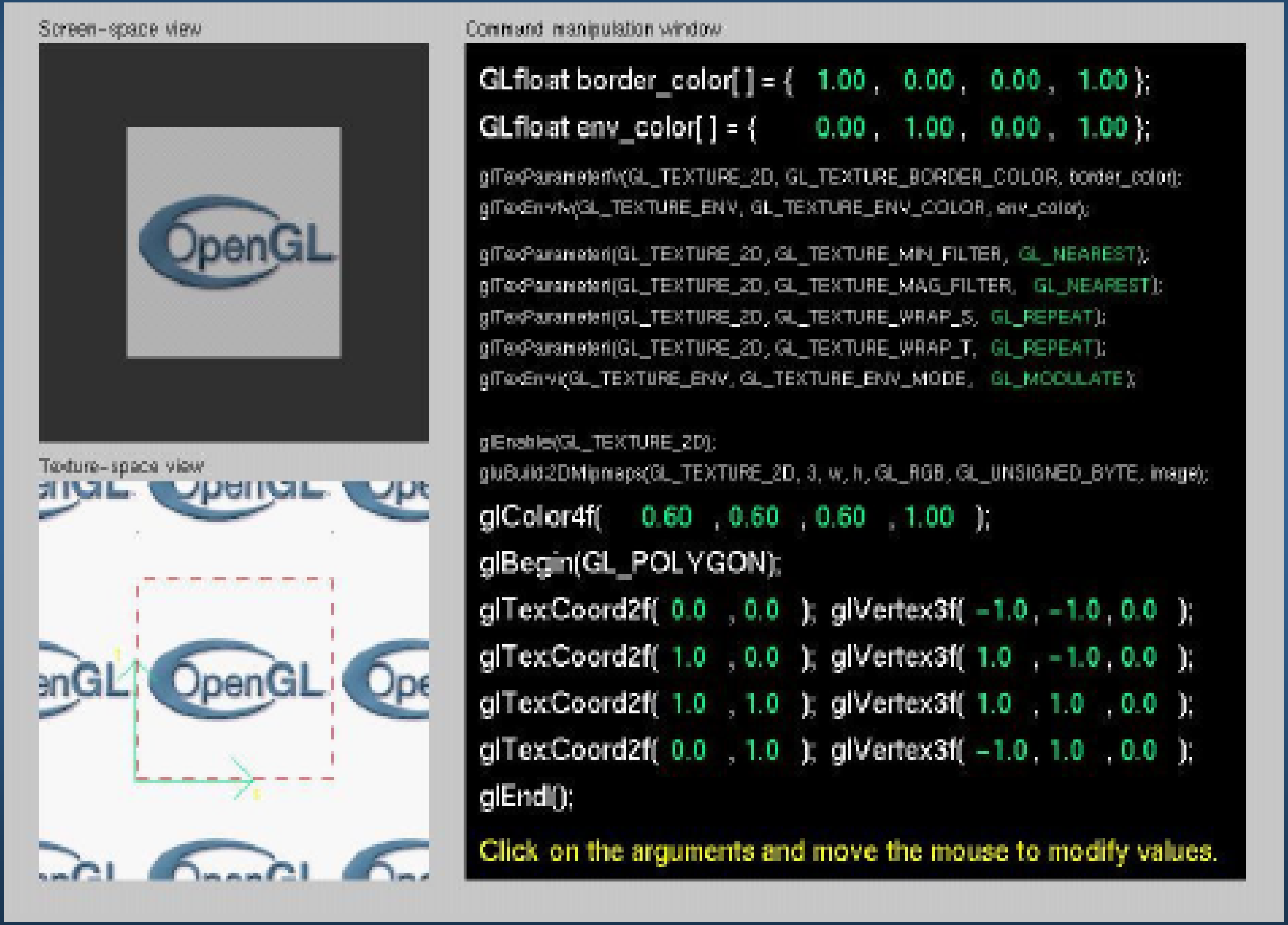
```
glCopyTexImage1D(...)
glCopyTexImage2D(...)
```

- Modify part of a defined texture

```
glTexSubImage1D(...)
glTexSubImage2D(...)
glTexSubImage3D(...)
```

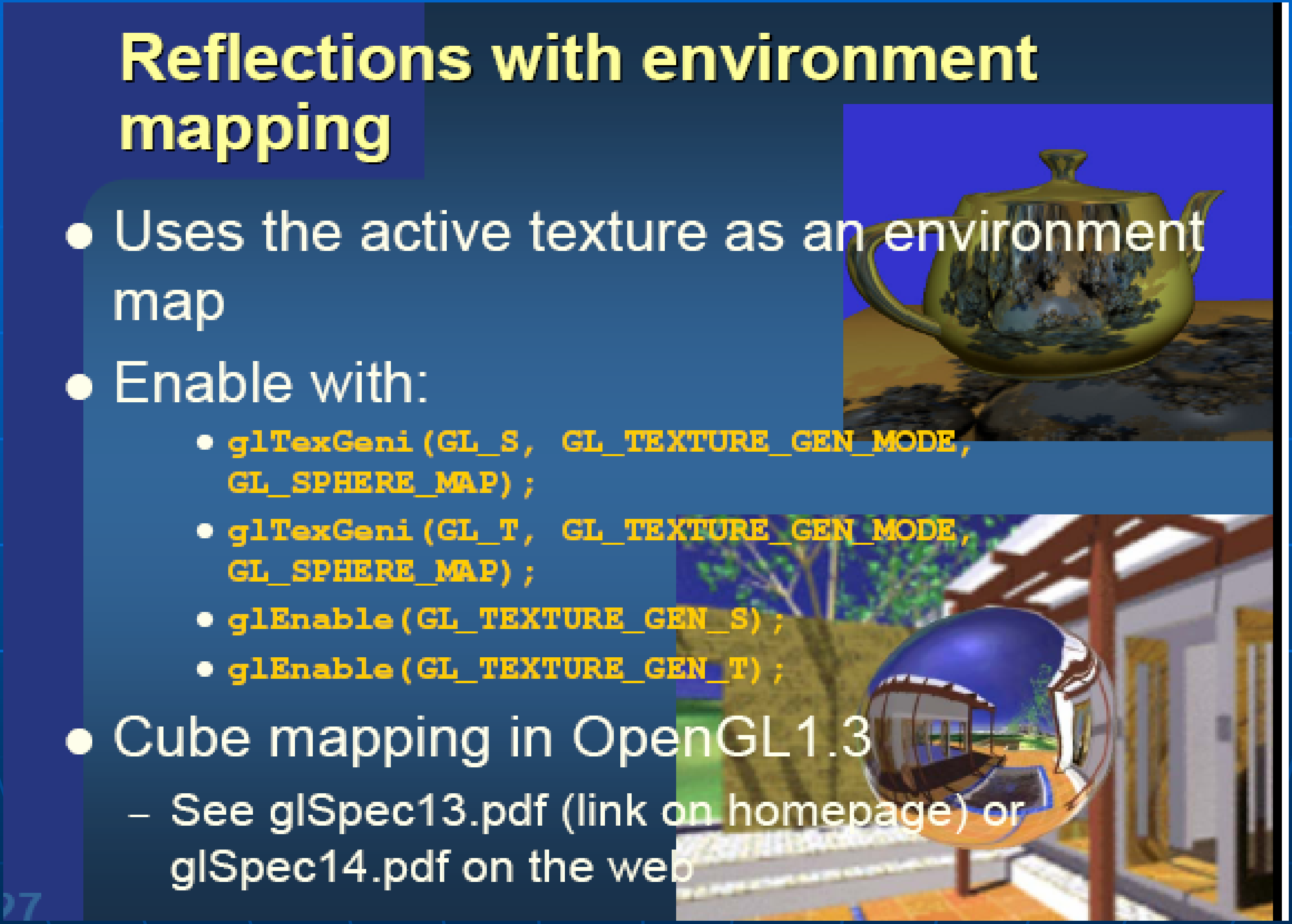
- Do both with *`glCopyTexSubImage2D(...)`*, etc.

Example of using texturing



```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,  GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,  GL_REPEAT);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  GL_MODULATE);


glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(   0.60  , 0.60  , 0.60  , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( 0.0  , 0.0  ); glVertex3f( -1.0 , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 0.0  ); glVertex3f( 1.0  , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 1.0  ); glVertex3f( 1.0  , 1.0  , 0.0  );
glTexCoord2f( 0.0  , 1.0  ); glVertex3f( -1.0 , 1.0  , 0.0  );
glEnd();
```

Click on the arguments and move the mouse to modify values.



Reflections with environment mapping

- Uses the active texture as an environment map
- Enable with:
  - `glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);`
  - `glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);`
  - `glEnable(GL_TEXTURE_GEN_S);`
  - `glEnable(GL_TEXTURE_GEN_T);`
- Cube mapping in OpenGL1.3
  - See glSpec13.pdf (link on homepage) or glSpec14.pdf on the web