

# Einführung in die Erweiterte Realität

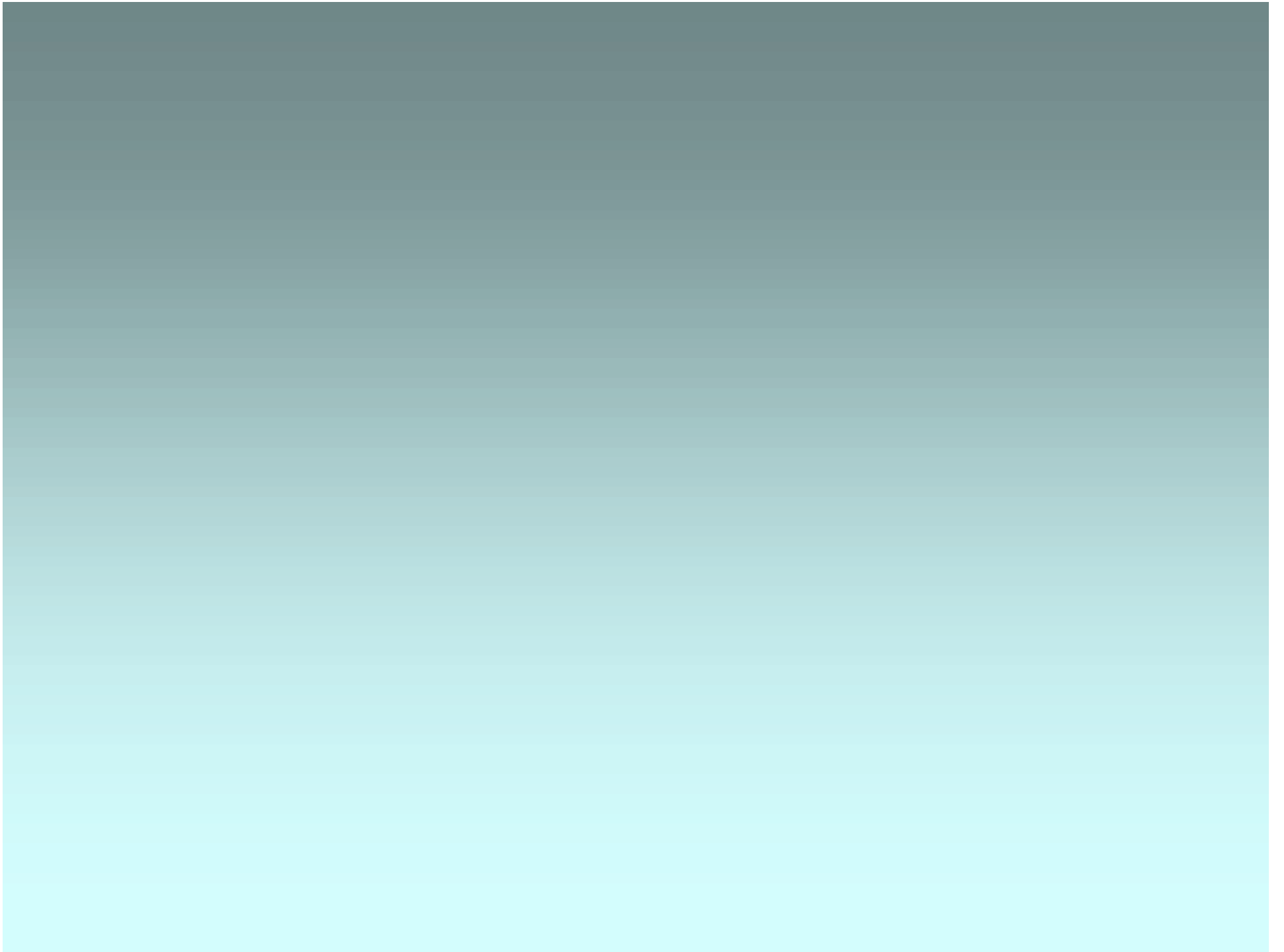
- 3. Open GL -

Gudrun Klinker

Nov 4, 2003

# Agenda

1. Object Representations in OpenGL
2. Transformations
3. Graphical State Variables
4. Combinations of Virtual and Real Objects



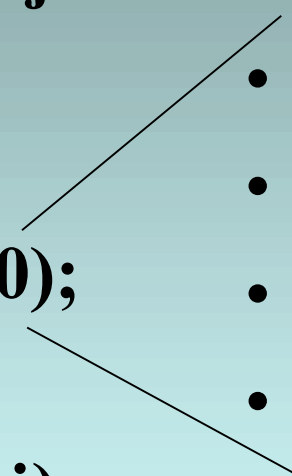
# 1. Object Representations in OpenGL

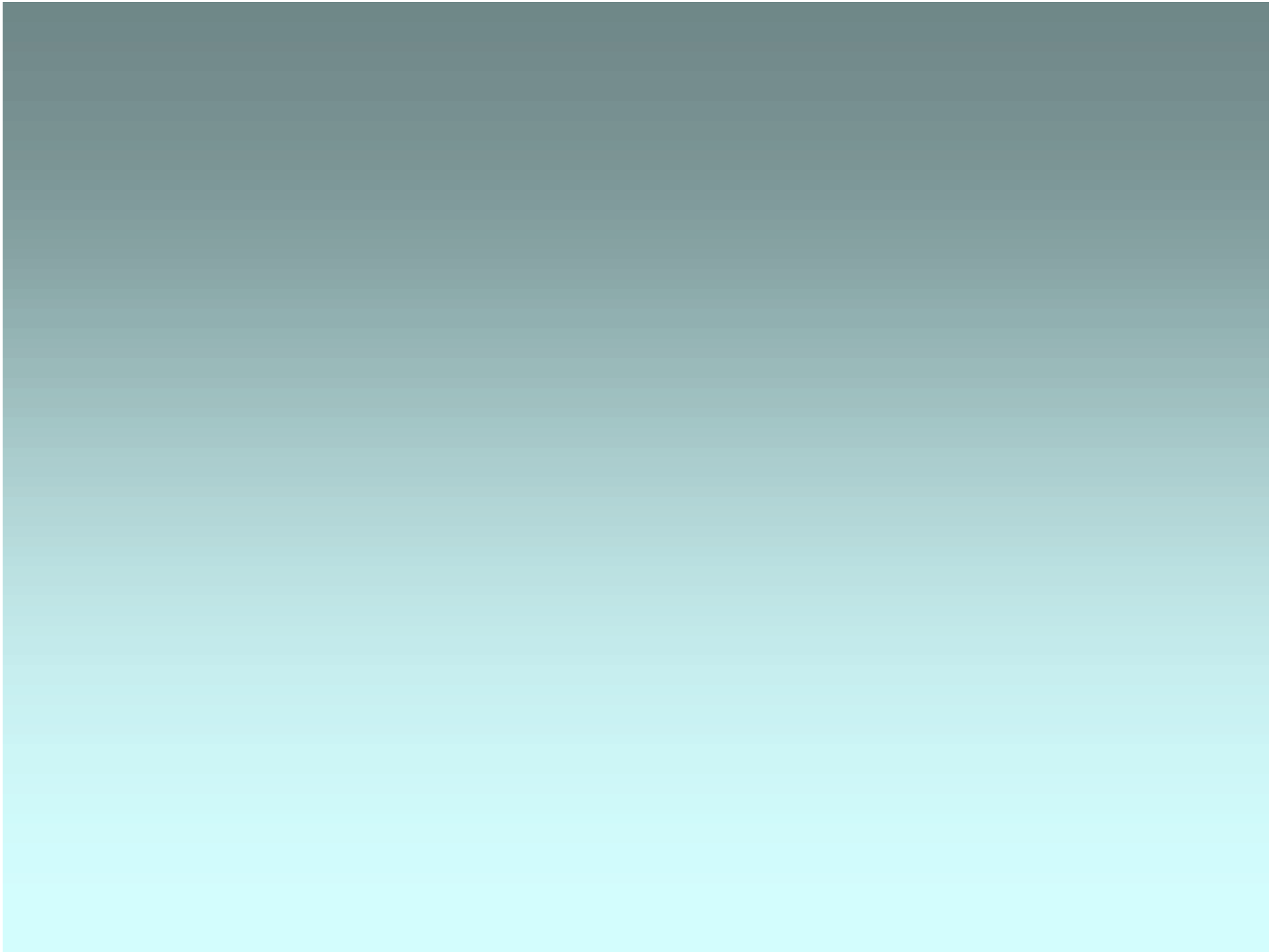
```
// draw a simple object (a yellow square)  
glColor3f (1.0, 1.0, 0.0);  
glBegin (GL_POLYGON);  
    glVertex3i (0, 0, 0);  
    glVertex3i (1, 0, 0);  
    glVertex3i (1, 1, 0);  
    glVertex3i (0, 1, 0);  
glEnd (...);
```

# glBegin (...)

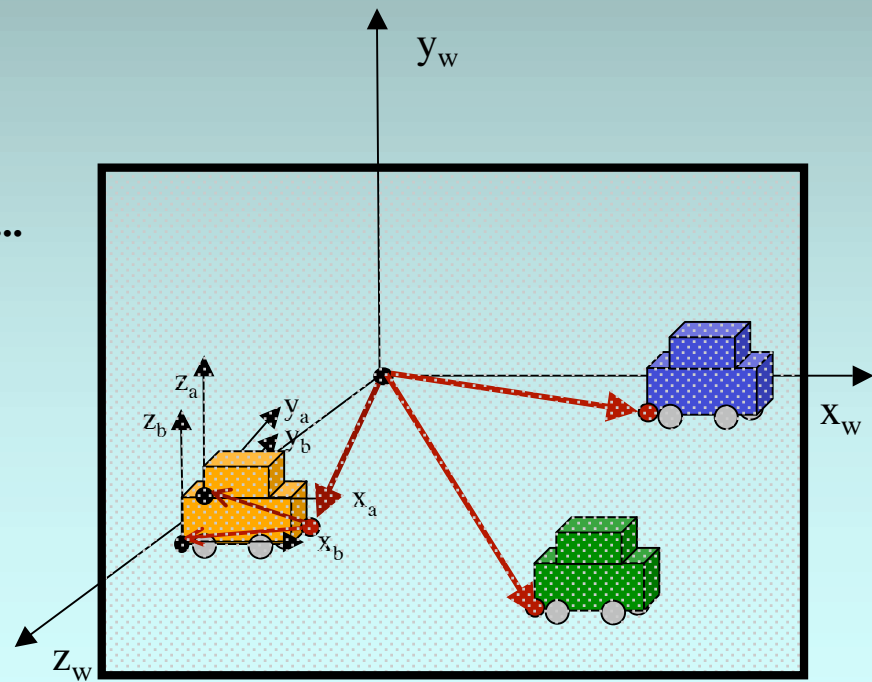
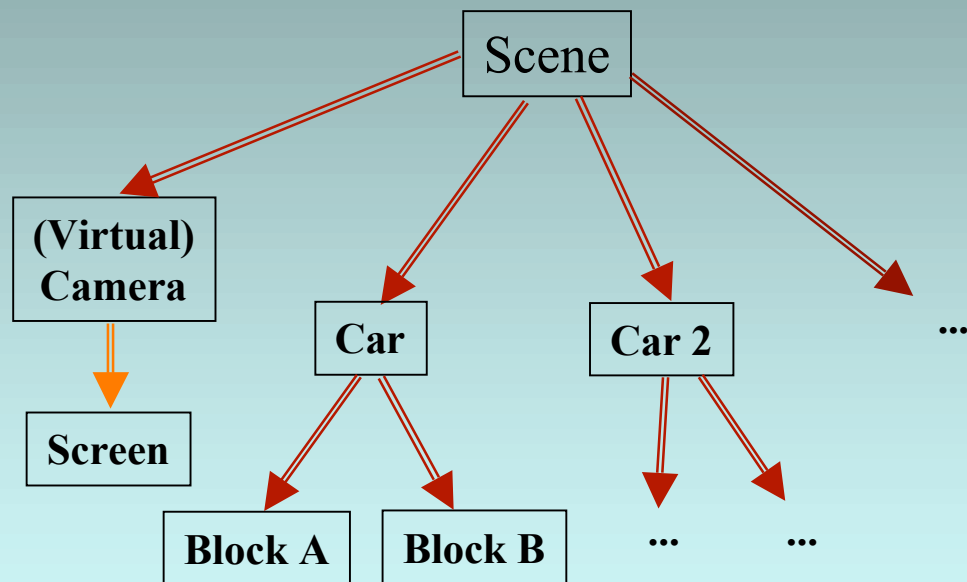
- **GL\_POINTS**
- **GL\_LINES**
- **GL\_LINE\_STRIP**
- **GL\_LINE\_LOOP**
- **GL\_TRIANGLES**
- **GL\_TRIANGLE\_STRIP**
- **GL\_TRIANGLE\_FAN**
- **GL\_QUADS**
- **GL\_QUAD\_STRIP**
- **GL\_POLYGON**

# More complex objects: Quadrics

- **GLUQuadricObj \*obj = gluNewQuadric();**
  - ...
  - **gluSphere(obj,r,20,20);**
  - ...
  - **gluDeleteQuadric(obj);**
- 
- A diagram consisting of two lines that originate from a single point on the right side of the slide. One line extends upwards and to the left, ending at the right edge of the first bullet point, 'gluNewQuadric()'. The other line extends downwards and to the left, ending at the right edge of the fourth bullet point, 'gluDeleteQuadric(obj);'. This diagram visually groups the first four items of the list as a single set of operations related to creating and deleting quadric objects.
- **gluCylinder(...)**
  - **gluDisk(...)**
  - **gluPartialDisk(...)**
  - **gluSphere(...);**



# Scene Graph





# 3D Transformations Using Homogeneous Coordinates

- Translation: `glTranslate*` ( $t_x, t_y, t_z$ )

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x + wt_x \\ y + wt_y \\ z + wt_z \\ w \end{bmatrix}$$

- Scaling: `glScale*` ( $s_x, s_y, s_z$ )

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix}$$

# 3D Transformations Using Homogeneous Coordinates

- Rotation:  $glRotate^*(\alpha, e_x, e_y, e_z)$

– around x

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x \\ \cos \alpha y - \sin \alpha z \\ \sin \alpha y + \cos \alpha z \\ w \end{bmatrix}$$

– around y

$$\begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \cos \alpha x + \sin \alpha z \\ y \\ -\sin \alpha x + \cos \alpha z \\ w \end{bmatrix}$$

– around z

$$\begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \cos \alpha x - \sin \alpha y \\ \sin \alpha x + \cos \alpha y \\ z \\ w \end{bmatrix}$$

# Composition of Transformations

- Initialize accumulative matrix  $C_0$  with identity matrix  $I$
- Post-multiply accumulative matrix  $C_i$  with transformation matrix  $M_i$  according to command.  $C_{i+1} := C_i M_i$
- Accumulated result represents complete transformation (independent of the number of transformation steps)
- Transformation sequence not commutative!!!

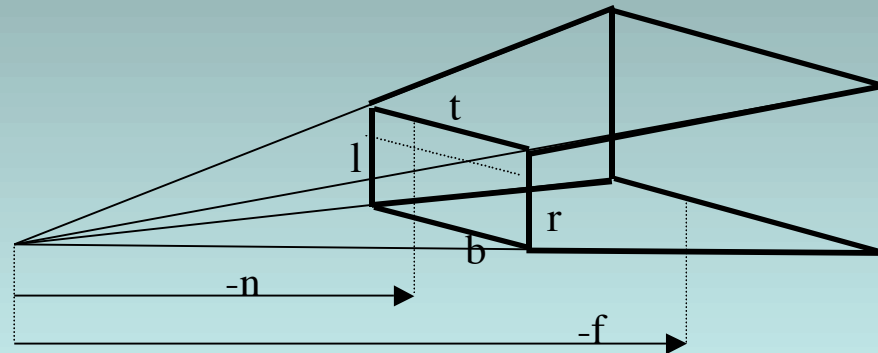
Example: translate1 - scale1 - rotate1 - translate2 - rotate2

$$I * T1 * S1 * R1 * T2 * R2 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

# 3D Projections

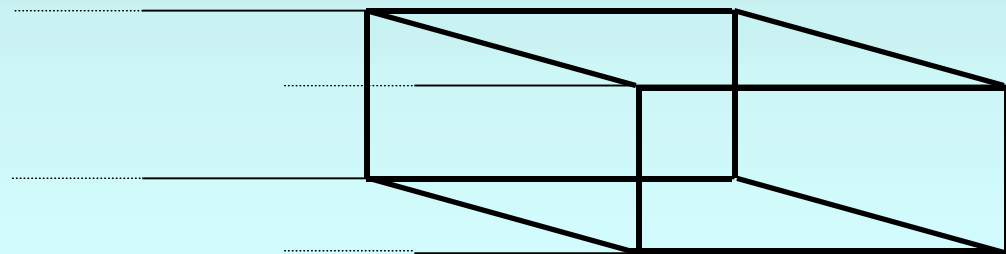
- Perspective Projection: `glFrustum*` (l,r,b,t,n,f)

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



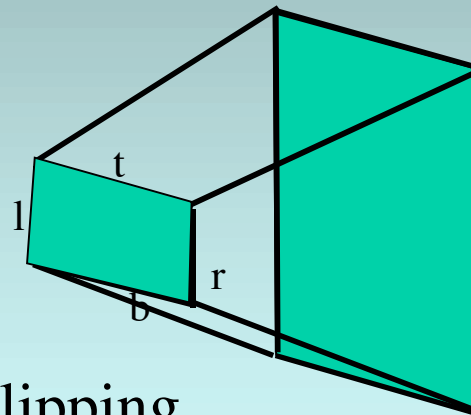
- Orthographic Projection: `glOrtho*` (l,r,b,t,n,f)

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# Viewport Transformation

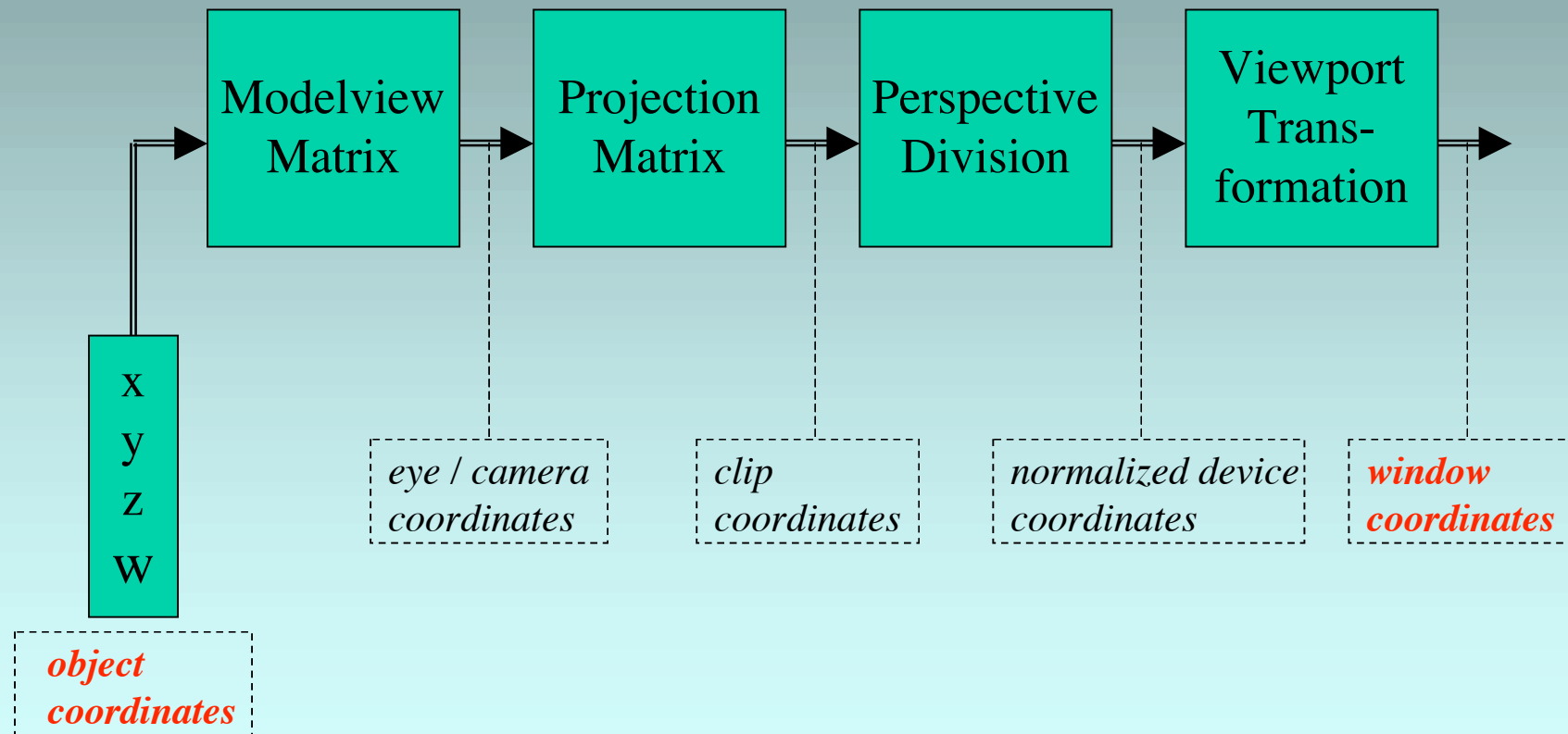
- `glViewport (x, y, width, height)`



window on  
the screen

near clipping  
plane of  
frustum

# Geometry Pipeline



# Sample Code

```
// Viewport Transformations
```

```
glViewport (0.0, 0.0, w, h);
```

```
// Projection Transformations
```

```
glMatrixMode (GL_PROJECTION);
```

```
glLoadIdentity ();
```

```
glFrustum (l,r,t,b,n,f); or gluPerspective (60.0, w/h, 1.0, 20.0);
```

```
// Viewing Transformations
```

```
glMatrixMode (GL_MODELVIEW);
```

```
glLoadIdentity ();
```

```
glTranslatef (0.0, 0.0, -5.0); // move world away from camera
```

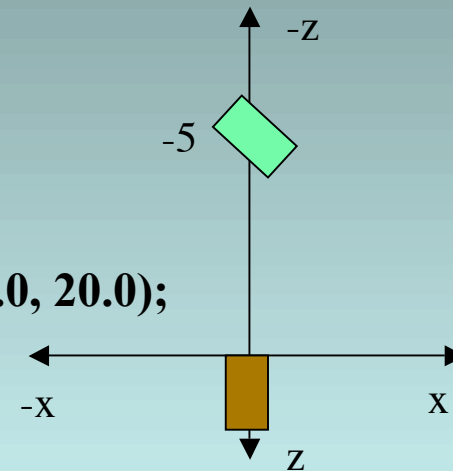
```
glRotatef (45.0, 0.0, 1.0, 0.0); // rotate world in front of the camera around y-axis
```

```
...
```

```
// Modeling Transformations
```

```
...
```

```
// Draw Object
```







### 3. “Hardware-Driven View”: Graphical State

Client Application  
(hardware driver)  
written in OpenGL



Special Graphics Hardware

- Registers for graphical state
  - draw styles, modes
  - draw colors
- Geometry pipeline
- Framebuffers
  - colored pixels (R,G,B)
  - alpha blending (□)
  - z-buffer
  - double buffering

# State Variables: Draw Modes

**glBegin(...)**

- **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**

**glEnd()**

# State Variables: Draw Colors and Styles

- for all:
  - Color (r,g,b): `glColor4f (1.0f,0.0f,0.0f,1.0f);`
- for Points:
  - Point size: `glPointSize(4.0f);`
- for Lines:
  - Line width: `glLineWidth (2.0f);`
  - Stipple patterns:  
`glEnable(GL_LINE_STIPPLE);`  
`glLineStipple (1,0xFFFF0)`

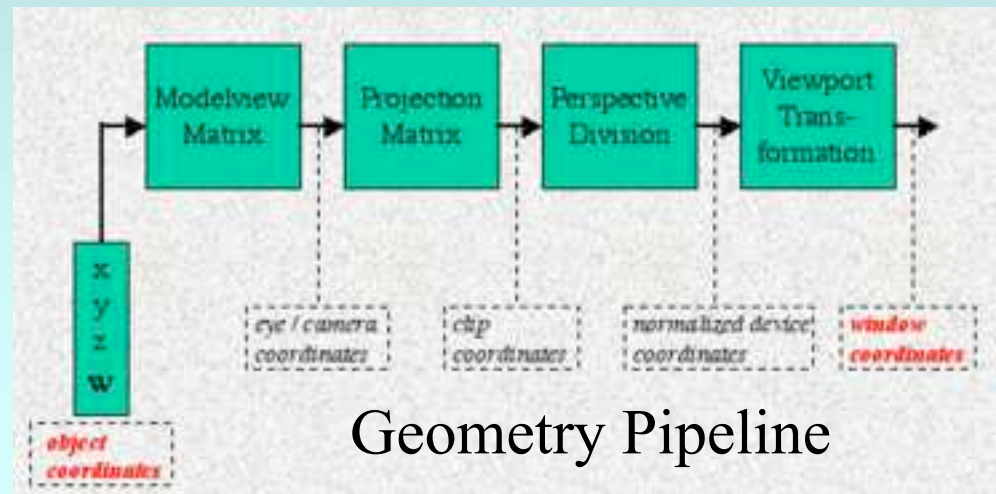
  
FFF 0 FFF 0 FFF 0 ...

# State Variables: Draw Styles (cont.)

- for Polygons:
  - **glPolygonMode (GL\_BACK, GL\_LINE);**
    - **Points, outlines, solids:**  
{GL\_POINT, GL\_LINE, GL\_FILL}
    - **Front (counterclockw.) and/or back:**  
{GL\_FRONT, GL\_BACK, GL\_FRONT\_AND\_BACK}
  - **Culling: glCullFace (GL\_FRONT);**
    - {GL\_FRONT, GL\_BACK}
  - **Stippling patterns (32x32 bitmaps)**
  - **Normal vectors**

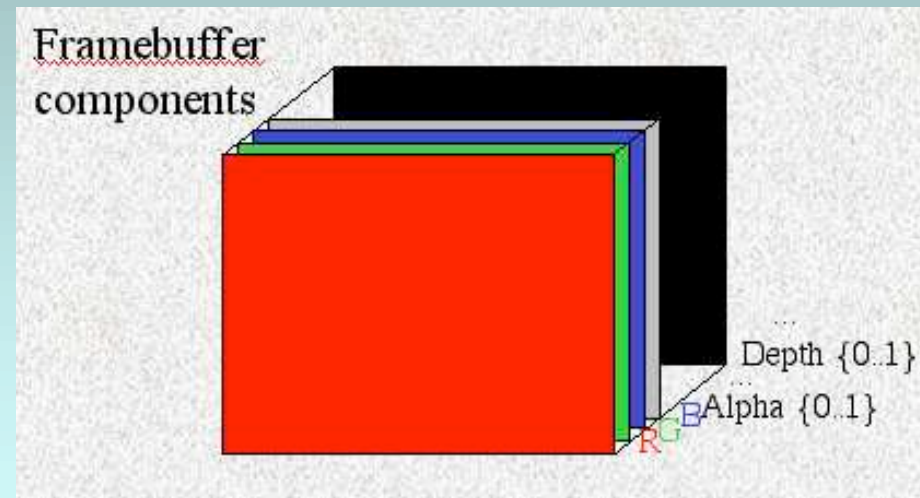
# State Variables: Transformation Matrices

- **glMatrixMode (GL\_MODELVIEW);**
  - {GL\_MODELVIEW, GL\_PROJECTIONS}
- **glLoadIdentity();**
- **glTranslate (t<sub>x</sub>, t<sub>y</sub>, t<sub>z</sub>);**
- **glRotate (e<sub>x</sub>, e<sub>y</sub>, e<sub>z</sub>, angle);**
- **glScale (s<sub>x</sub>, s<sub>y</sub>, s<sub>z</sub>);**
- ...



# State Variables: Lighting, Textures, Framebuffer

- **Lighting**
- **Texture Mapping**
- **Framebuffer Control**
  - **RGBA, depth, stencil, accumulation**
  - **Doublebuffering, stereo**



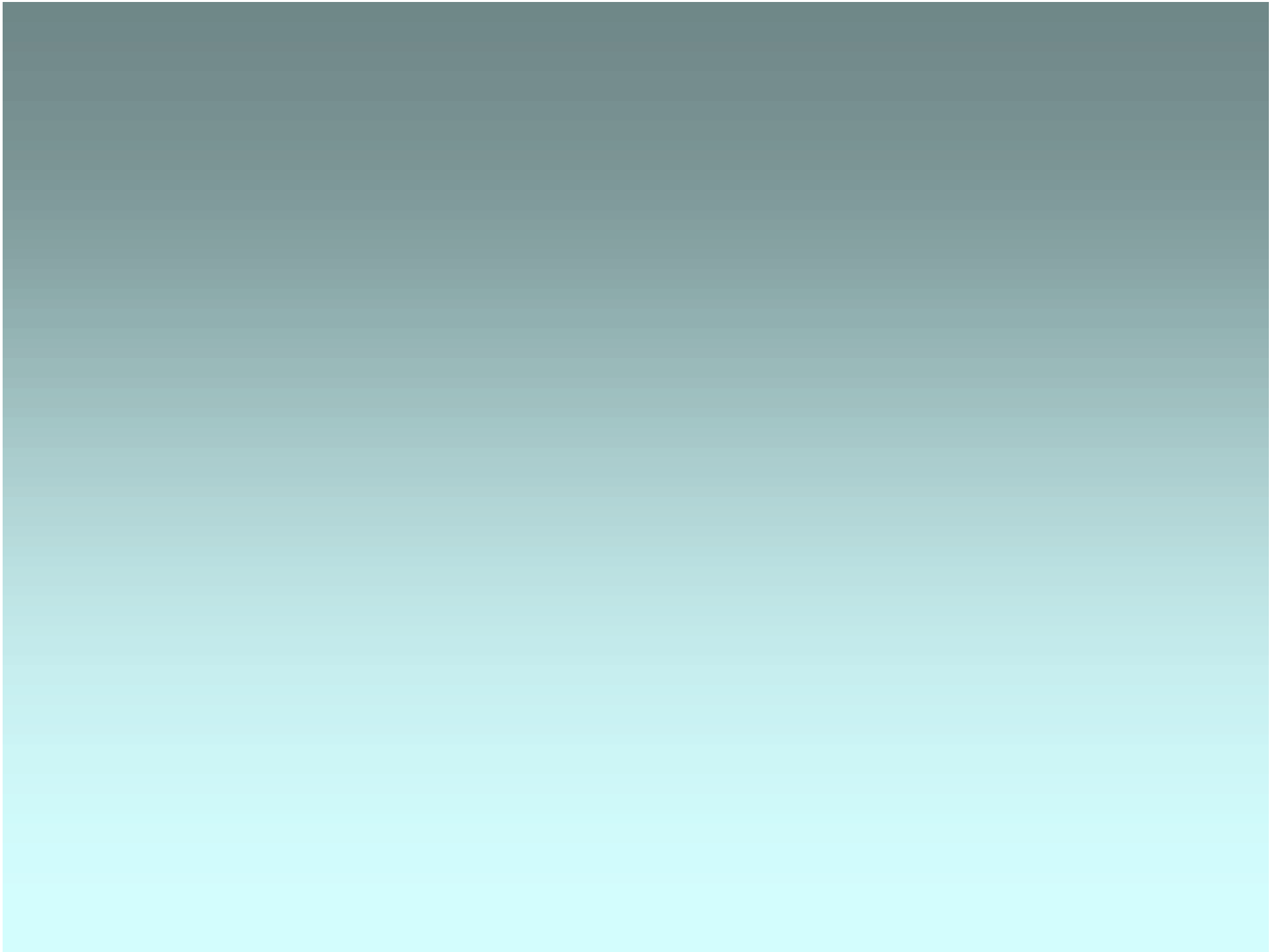
# Controlling the Grafical State

- Communication
  - `glFlush()`
- Functionality
  - `glEnable(...), glDisable(...)`
  - `glPushAttrib(...), glPopAttrib(...)`
  - `glPushMatrix(...), glPopMatrix(...)`
- Querying the graphical state
  - `glGetIntegerv(...)`
  - `glGetError(...)`
- Framebuffer control
  - `glClear(...), glClearColor(...), glClearDepth(...), ...`
  - `glXSwapBuffers()`

# Example

```
Main () {  
    // ... open a window in double buffer mode (depends on window system)...  
    glClearColor (0.0, 0.0, 0.0, 0.0); // black  
    glClearDepth (0.0);  
    glEnable (GL_DEPTH_TEST);  
    glDepthFunc (GL_LEQUAL);  
    do {  
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
        // ... coordinate transformations (viewing information)  
        glPushMatrix(); glPushAttrib(...);  
        // ... draw objects  
        glPopMatrix(); glPopAttrib(...);  
        // ... clean up  
        glFlush();  
        glXSwapBuffers (...);  
    } while (TRUE); // infinite loop  
}
```





## 4. Occlusions between Real and Virtual Objects

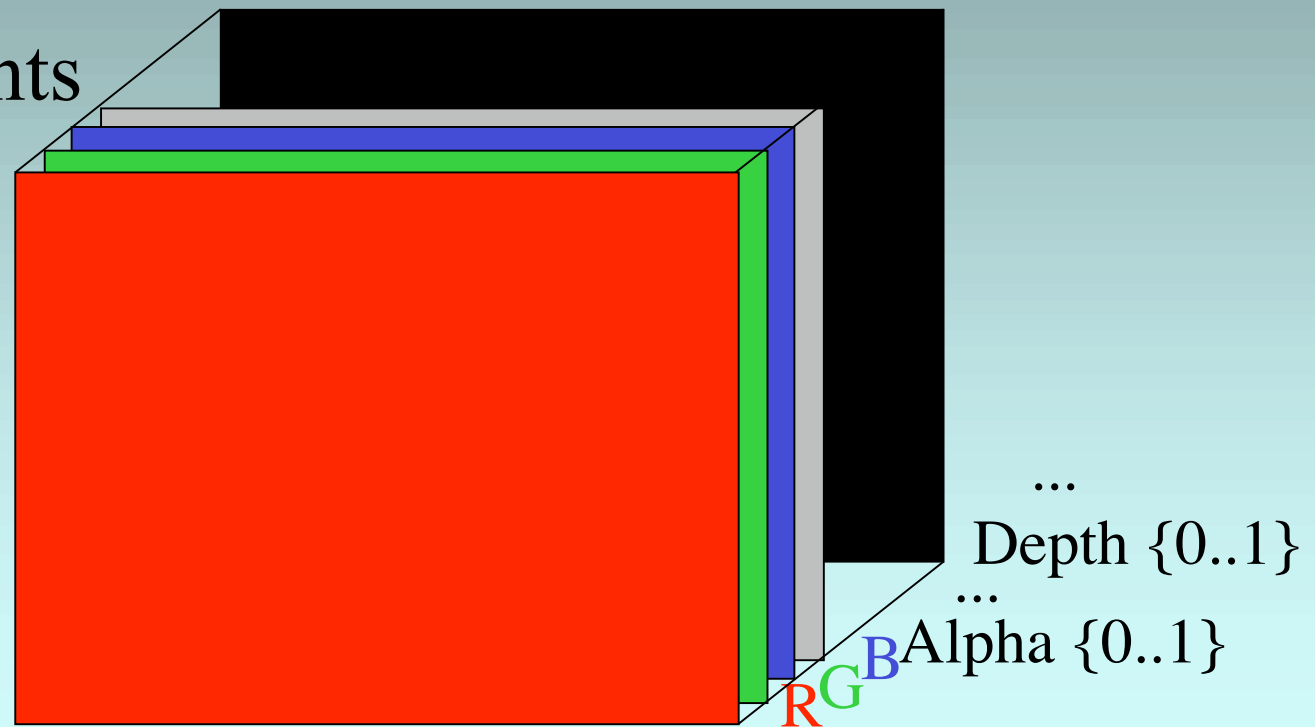
In Front In Back	Real Objects	Virtual Objects
Real Objects	optical occlusion	
Virtual Objects		

# Occlusions between Real and Virtual Objects

In Front In Back	Real Objects	Virtual Objects
Real Objects	optical occlusion	
Virtual Objects		graphics hardware (z-buffering)

# Hidden Surface Removal Using a Z-Buffer

- Framebuffer components



# Depth Buffer Manipulation

- **glClearDepth (1.0);**
- **glClear (GL\_DEPTH\_BUFFER\_BIT);**
- **glEnable (GL\_DEPTH\_TEST);**
- **glDepthFunc (GL\_LEQUAL);**
- **glDepthMask (GL\_TRUE);**

# Occlusions between Real and Virtual Objects

In Front In Back	Real Objects	Virtual Objects
Real Objects	optical occlusion	graphics in front of video
Virtual Objects		graphics hardware (z-buffering)

# Drawing in the 2D image plane

- Orthographic view

```
glOrtho (0,imgwidth, 0,imgheight, -1.0,1.0);
```

- Initialize modelling/viewing transformation

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

- Flip image (image array starts at upper left corner, OpenGL starts at lower left corner)

```
glPixelZoom(1.0, -1.0);
```

- Define origin of image coordinate system:

```
glRasterPos2i (0, imgheight);
```

# Drawing 2D images and marks

- **Draw image (upper left corner at raster pos)**  
**glDrawPixels (imgwidth, imgheight, GL\_RGBA,  
GL\_UNSIGNED\_BYTE, data);**
- **Draw 2D marks (stored in a display-list) onto the image**  
**glCallList (MyMarkers);**

```
glNewList (MyMarkers, GL_COMPILE);  
    glBegin (GL_LINES);  
        glVertex2d (x, imgheight-y);  
        ...  
    glEnd ();  
glEndList ();
```





# Occlusions between Real and Virtual Objects

In Front In Back	Real Objects	Virtual Objects
Real Objects	optical occlusion	graphics in front of video
Virtual Objects	<b>video + transparent reality models + opaque virtual models</b>	graphics hardware (z-buffering)

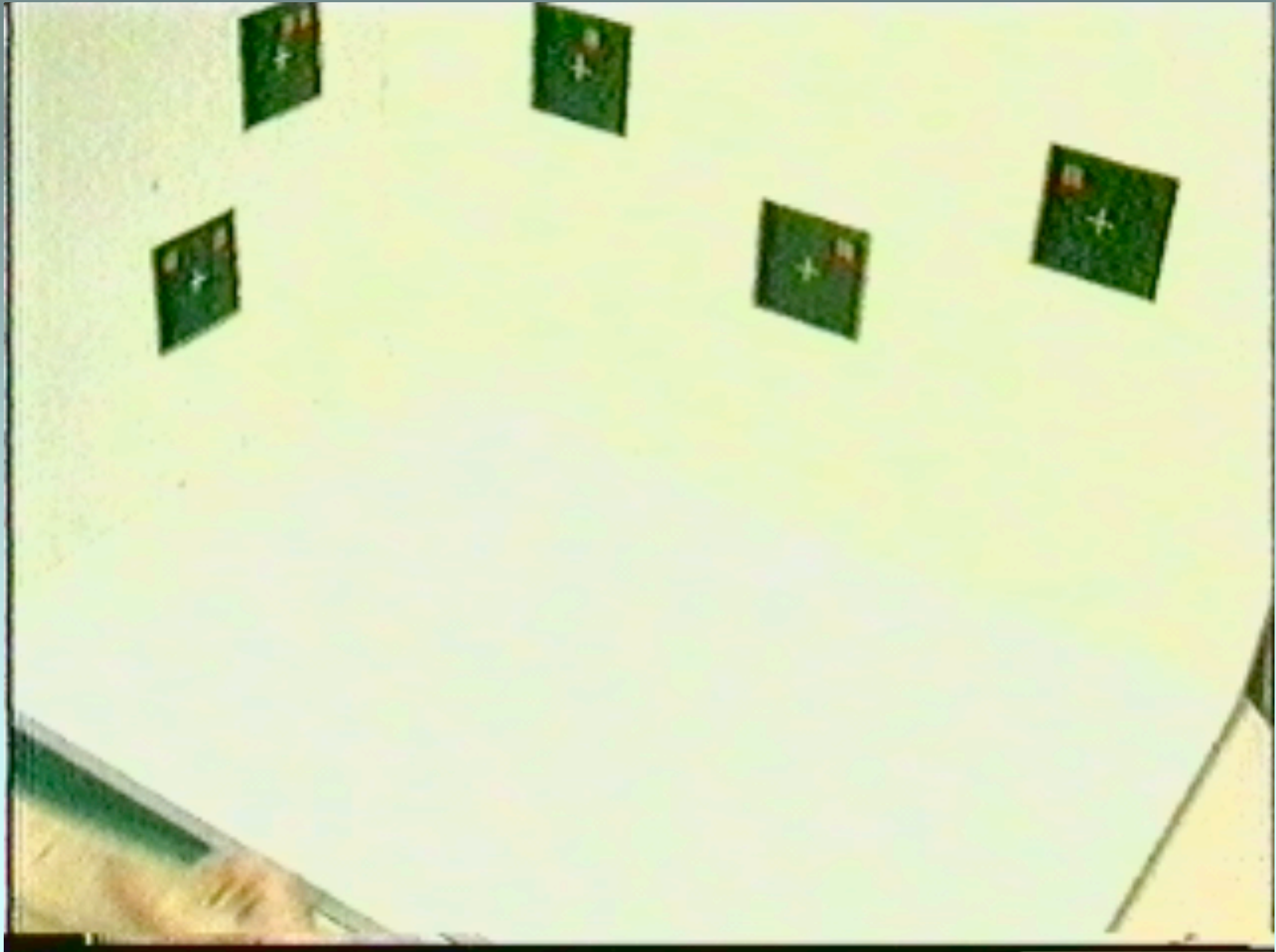
# Integrating Virtual Objects into a Real Scene

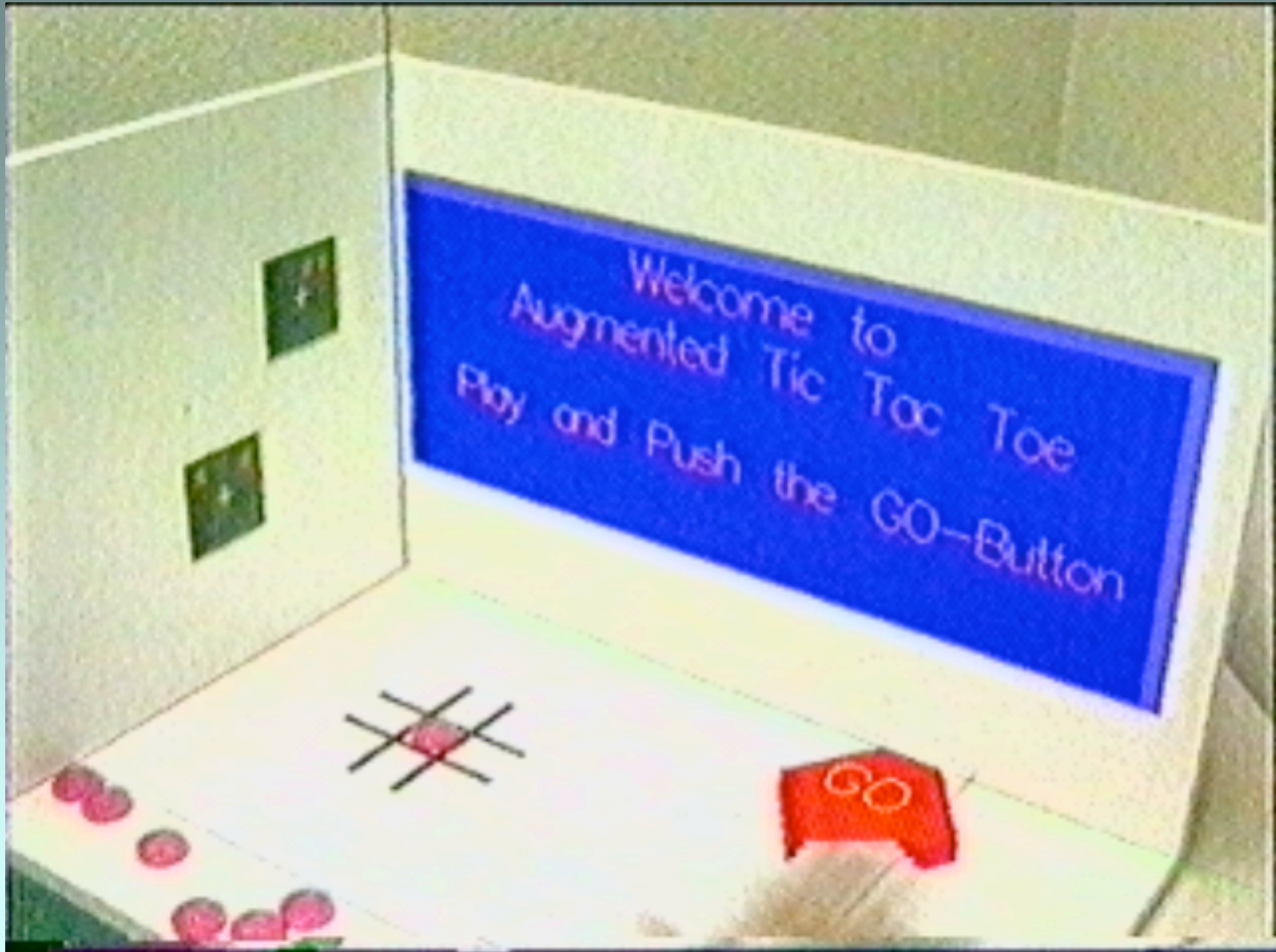
- Prerequisites:
  - Obtain geometrically precise models of all real objects.
  - Determine the current position of mobile objects.
- For a given camera position:
  - Draw video image (as background).
  - Initialize depth buffer.
  - Draw transparent models of real objects.
  - Draw opaque models of virtual objects.

# Drawing Transparent Pixels

## - Color Mask -

- **...Draw video image**
- **...Initialize depth buffer**
- **glColorMask (GL\_FALSE, GL\_FALSE, GL\_FALSE, GL\_FALSE);**
- **...Draw reality model**
- **glColorMask (GL\_TRUE, GL\_TRUE, GL\_TRUE, GL\_TRUE);**
- **...Draw virtual model**







# Blending Factors

- **GL\_ZERO** (0, 0, 0, 0)
- **GL\_ONE** (1, 1, 1, 1)
- **GL\_DST\_COLOR** ( $R_d$ ,  $G_d$ ,  $B_d$ ,  $A_d$ )
- **GL\_SRC\_COLOR** ( $R_s$ ,  $G_s$ ,  $B_s$ ,  $A_s$ )
- **GL\_ONE\_MINUS\_DST\_COLOR** ( $1-R_d$ ,  $1-G_d$ ,  $1-B_d$ ,  $1-A_d$ )
- **GL\_ONE\_MINUS\_SRC\_COLOR** ( $1-R_s$ ,  $1-G_s$ ,  $1-B_s$ ,  $1-A_s$ )
- **GL\_SRC\_ALPHA** ( $A_s$ ,  $A_s$ ,  $A_s$ ,  $A_s$ )
- **GL\_ONE\_MINUS\_SRC\_ALPHA** ( $1-A_s$ ,  $1-A_s$ ,  $1-A_s$ ,  $1-A_s$ )
- **GL\_DST\_ALPHA** ( $A_d$ ,  $A_d$ ,  $A_d$ ,  $A_d$ )
- **GL\_ONE\_MINUS\_DST\_ALPHA** ( $1-A_d$ ,  $1-A_d$ ,  $1-A_d$ ,  $1-A_d$ )
- **GL\_SRC\_ALPHA\_SATURATE** ( $f$ ,  $f$ ,  $f$ , 1)  $f = \min(A_s, 1-A_d)$



# Examples of blending combinations

- **GL\_SRC\_ALPHA, GL\_SRC\_ALPHA**; alpha = 0.5
- **GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA**; alpha = 0.25
- **GL\_SRC\_ALPHA, GL\_ONE**; alpha =  $1/n$  (for n images)
- **GL\_ZERO, GL\_ONE** (for transparent models of real objects)

# Drawing Transparent Pixels

## - Alpha Blending -

- **...Draw video image**
- **...Initialize depth buffer**
- **glEnable (GL\_BLEND);**
- **glBlendFunc (sfactor=0, dfactor=1);**
- **...Draw reality model**
- **glBlendFunc (sfactor=1, dfactor=0);**
- **...Draw virtual model**



