Einführung in die Erweiterte Realität

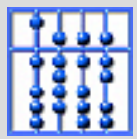
3. OpenGL,
Mixing Virtual and Real Objects

Prof. Gudrun Klinker, Ph.D.
Institut für Informatik,Technische Universität München
klinker@in.tum.de

Nov 9, 2004

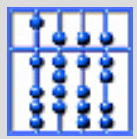
Literature on the web

- http://www.opengl.org/
- http://www.opengl.org/documentation/index.html
  - redbook.pdf
- http://www.opengl.org/resources/tutorials/
- http://www.opengl.org/resources/tutorials/overview.html

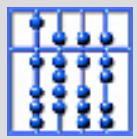


Agenda

1. Event Loop and Window Management
2. Object Representations in OpenGL
3. Transformations
4. Graphical State Variables
5. Combinations of Virtual and Real Objects

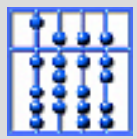


1. Event Loop and Window Management



Event Loop

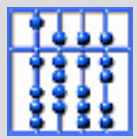
```
Main () {
    // … some initializations:
    // - open a window
    // - initialize hardware on graphics card
    // - initialize state variables (sw) in OGL driver

    do {  // EVENT LOOP
      // … loop initializations
      // … get events from windowing system (mouse, keyboard, etc)
      // … setup coordinate systems
      // … draw objects
      // … flush pipeline, swap buffers etc
      // … clean up
    }  while (TRUE);   // infinite loop
}
```

GLUT: The OpenGL Utility Toolkit

- Communication with windowing systems to
  - Open, initialize, resize, … windows
  - Receive input events from the windows
  - Draw into the windows

  - Routines to draw more complex objects

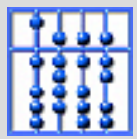


Window management with GLUT

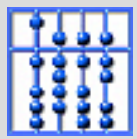
```
int main (int argc, char** argv) {
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (250,250);
   glutInitWindowPosition (100,100);
   glutCreateWindow (“Hello”);
   init();   // our own initializations
   glutDisplayFunc(display);

   glutMainLoop();
   return 0;
}

void init (void)   {}
void display (void) {}
```

Event management with GLUT

```
// event management
void glutDisplayFunc (void (*func) (void))
void glutReshapeFunc (void (*func)(int width, int height))
void glutKeyboardFunc (void (*func)(unsigned int key, int x, int y))
void glutMouseFunc (void (*func)(int button, int state, int x, int y))
void glutMotionFunc (void (*func)(int x, int y))
void glutPostRedisplay (void (*func)(void))

// for animations
void glutIdleFunc(void (*func) (void))

// main loop (infinitely running)
void glutMainLoop (void)
```

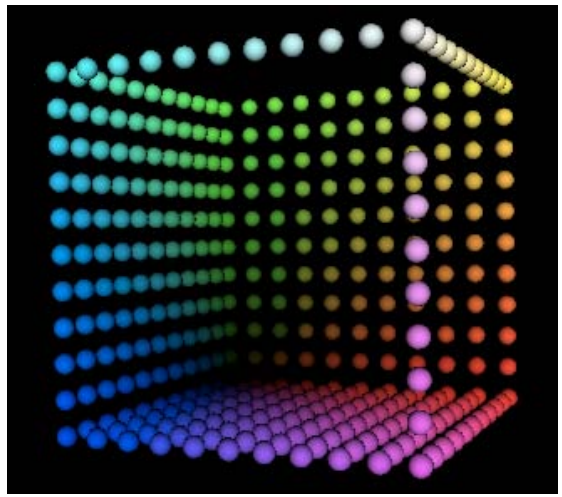



2. Object Representations in Open GL



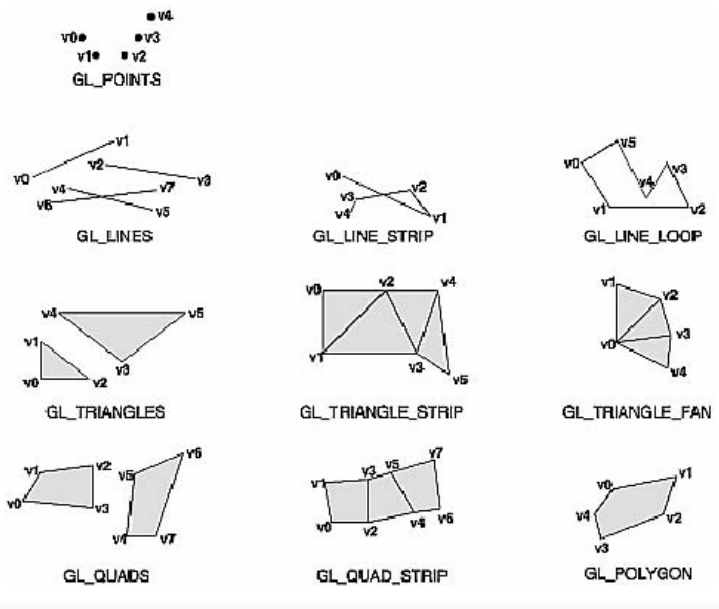
Object Representations in OpenGL

```
// 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 (…);
```



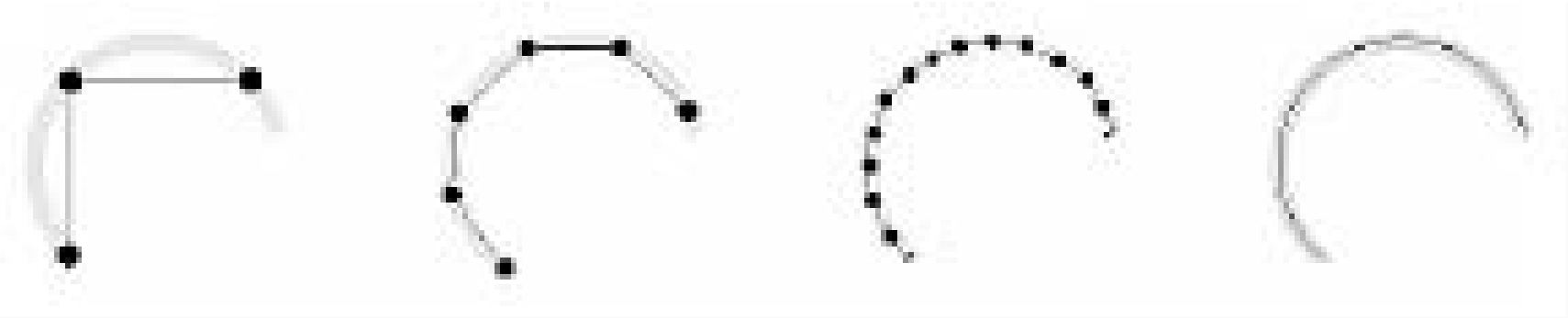
glBegin (…)

- **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

- Approximation of a circle by a multi-edged polygon

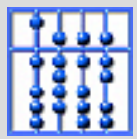


More complex objects: Quadrics

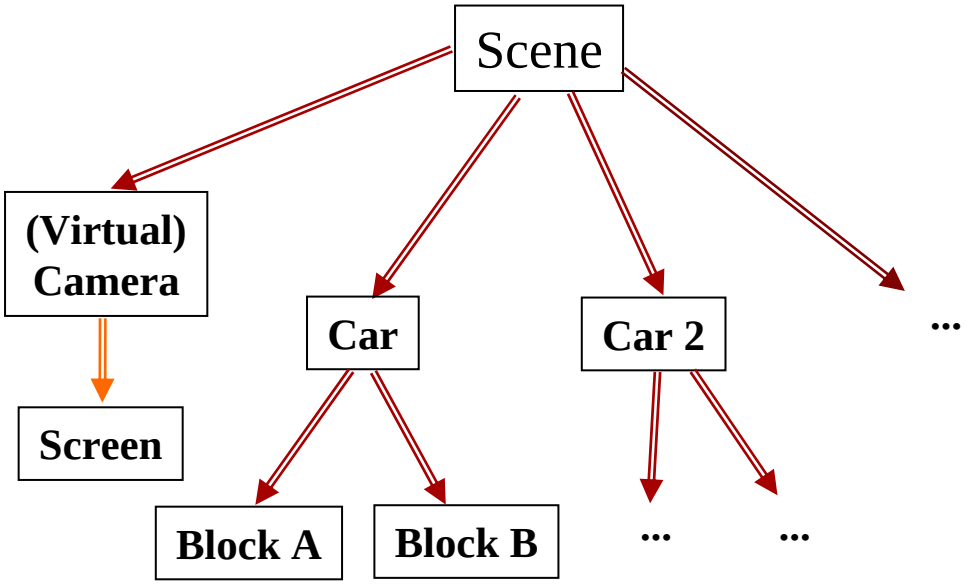
- **GLUQuadricObj *obj = gluNewQuadric();**
- **…**
- **gluSphere(obj,r,20,20);**
- **...**
- **gluDeleteQuadric(obj);**

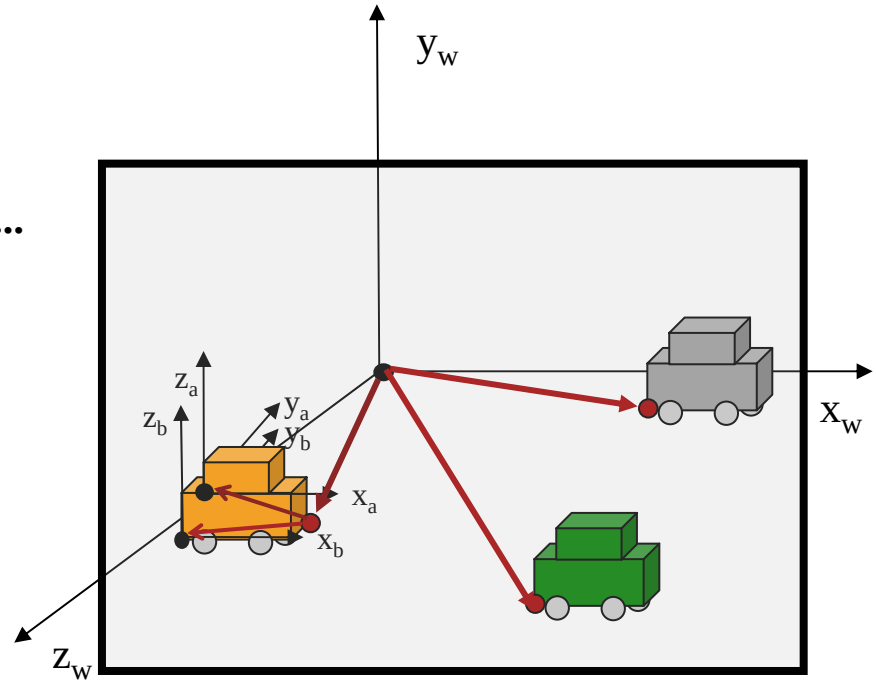
- **gluCylinder(…)**
- **gluDisk(…)**
- **gluPartialDisk(…)**
- **gluSphere(…);**

3. Transformations



Scene Graph


Scene
(Virtual) Camera
Screen
Car
Car 2
...
Block A
Block B
...
...


y_w
x_w
z_w
z_a
y_a
z_b
y_b
x_a
x_b

3D Transformations Using Homogeneous Coordinates

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

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

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

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

3D Transformations Using Homogeneous Coordinates

- Rotation: glRotate* (α, e_x, e_y, e_z)
  - around x
  - around y
  - around z

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

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

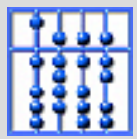
$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \cos\alpha\, x - \sin\alpha\, y \\ \sin\alpha\, x + \cos\alpha\, y \\ z \\ w \end{bmatrix}$$

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

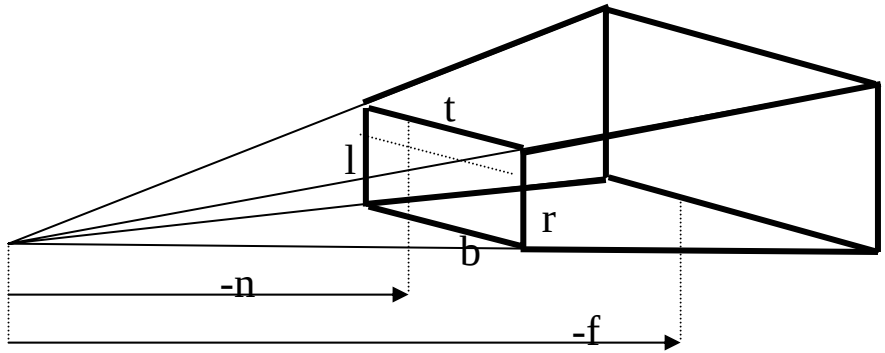
$$I * T1 * S1 * R1 * T2 * R2 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$



3D Projections

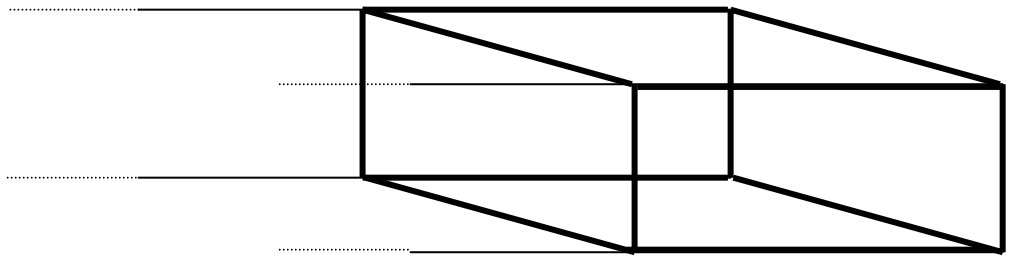
- Perspective Projection: glFrustum* (l,r,b,t,n,f)

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



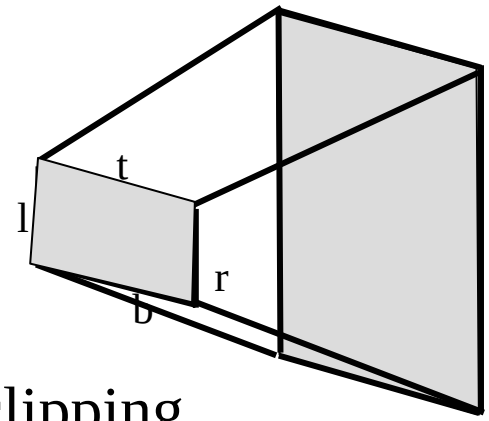
- Orthographic Projection: glOrtho* (l,r,b,t,n,f)

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



Viewport Transformation

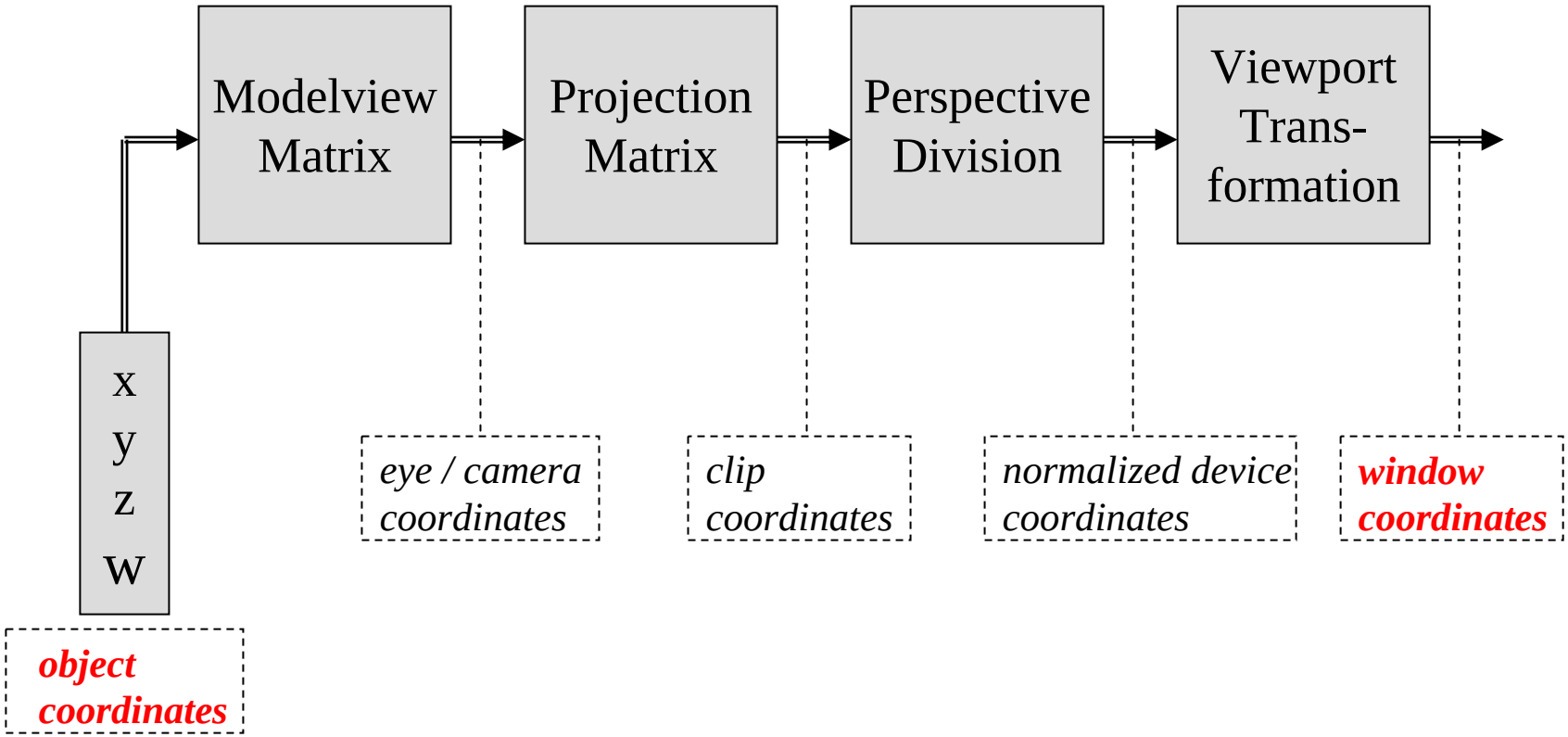
- glViewport (x, y, width, height)



window on the screen

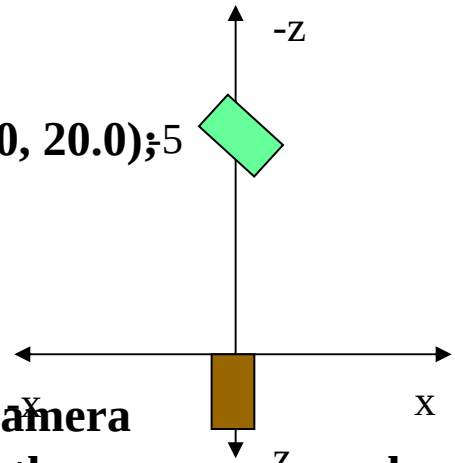
near clipping plane of frustum

Geometry Pipeline

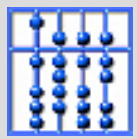


Sample Code

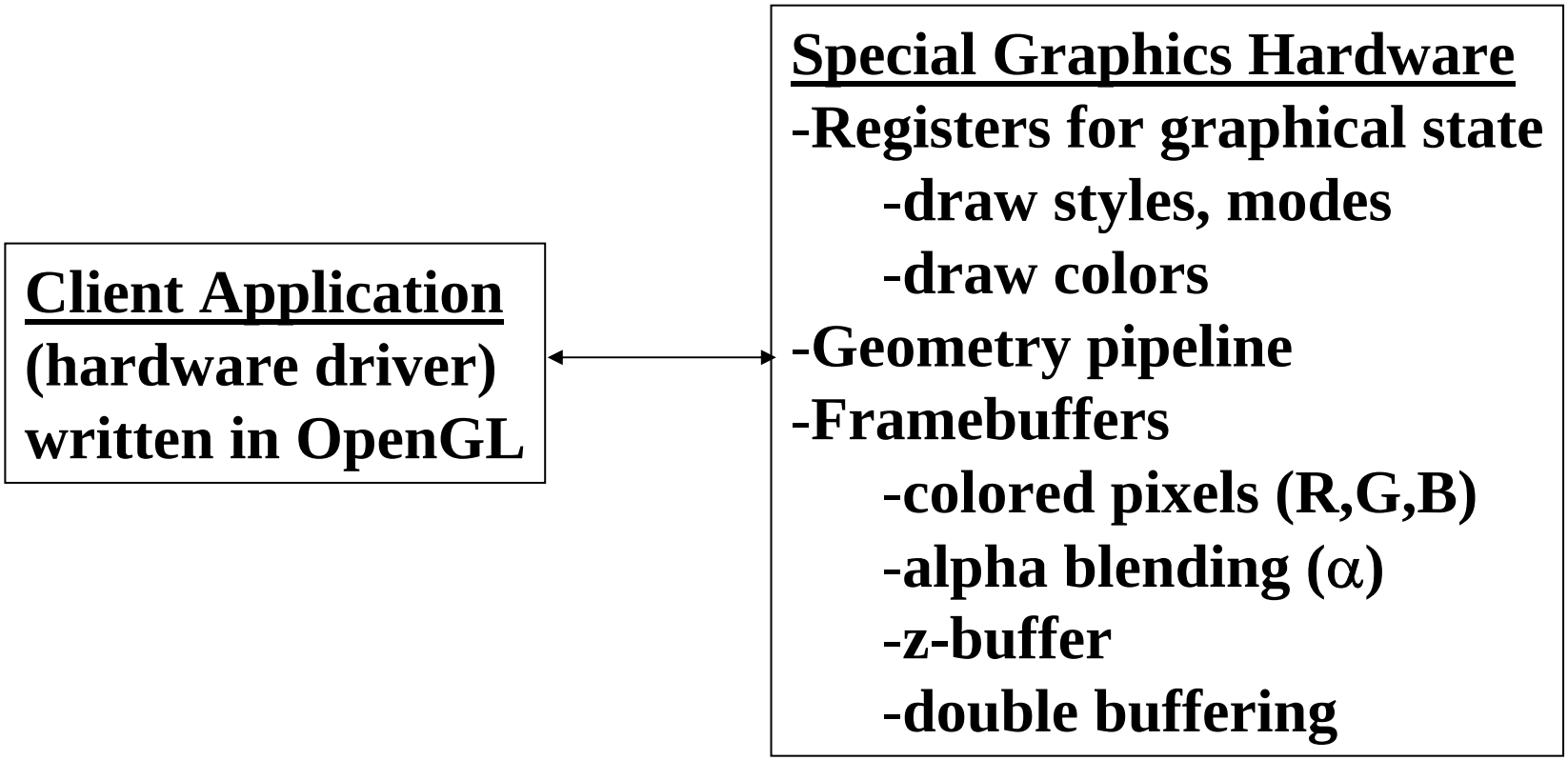
```
// Viewport Transformations
glViewport (0.0, 0.0, w, h);

// 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
…
```



4. Graphical State



“Hardware-Driven View”

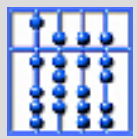


State Variables: - Draw Modes -

glBegin(…)

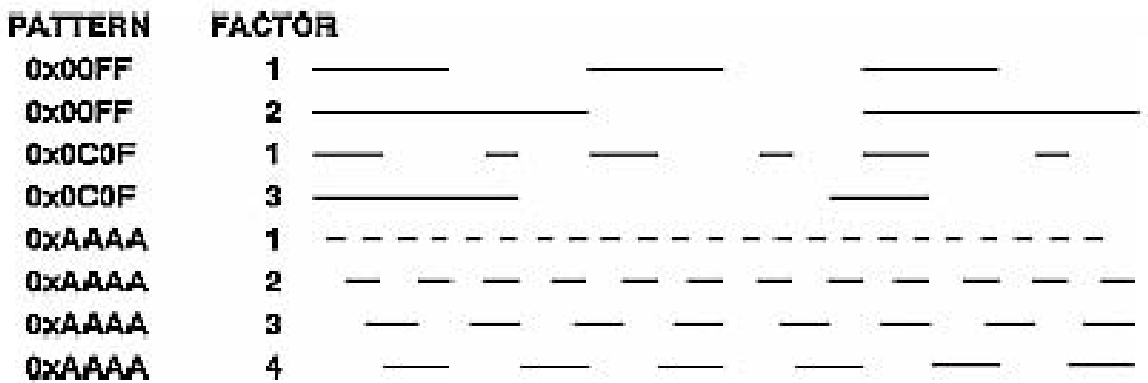
- **GL_POINTS**
- **GL_LINES**
- **GL_LINE_STRIP**
- **GL_LINE_LOOP**
- **GL_TRIANGLES**
- **GL_TRIANGLE_STRIP**
- **GL_TRIANGLE_FAN**
- **GL_QUADS**
- **GL_QUAD_STRIP**
- **GL_POLYGON**

glEnd()



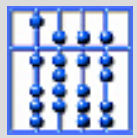
State Variables: - Draw Colors and Styles -

- 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 (1,0xFFF0)



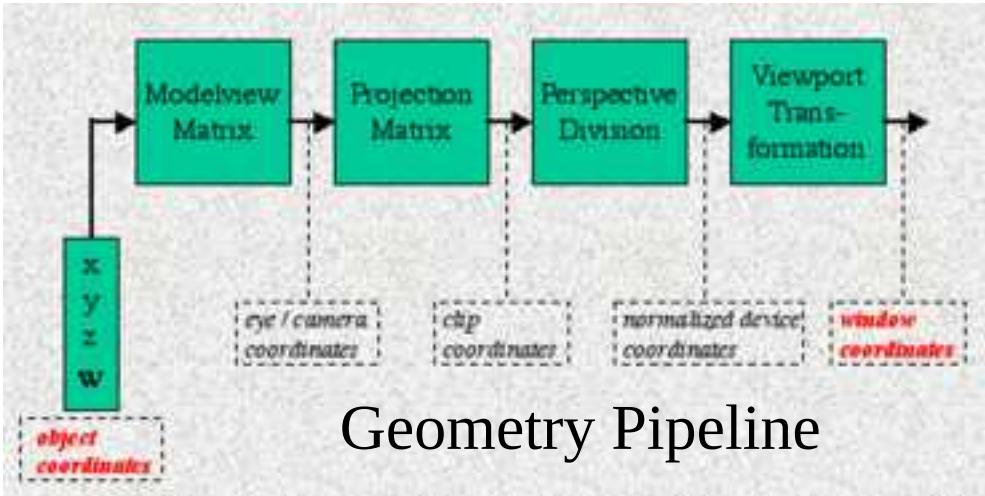
State Variables: - Draw Styles (cont.) -

- 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 -

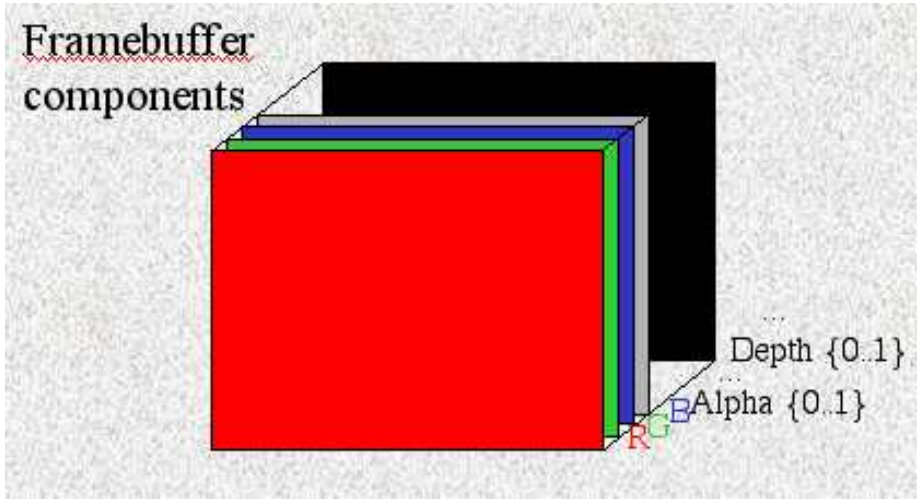
- **glMatrixMode (GL_MODELVIEW);**
  - **{GL_MODELVIEW, GL_PROJECTIONS}**
- **glLoadIdentityMatrix();**
- **glTranslate (t_x, t_y, t_z);**
- **glRotate (e_x, e_y, e_z, angle);**
- **glScale (s_x, s_y, sz);**
- **…**



Geometry Pipeline

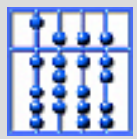
State Variables: - Lighting, Textures, Framebuffer -

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



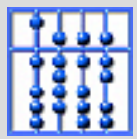
Controlling the Grafical State

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

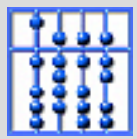


Example

```
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
}
```

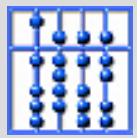



5. Combinations of Real and Virtual Objects



Occlusions between Real and Virtual Objects

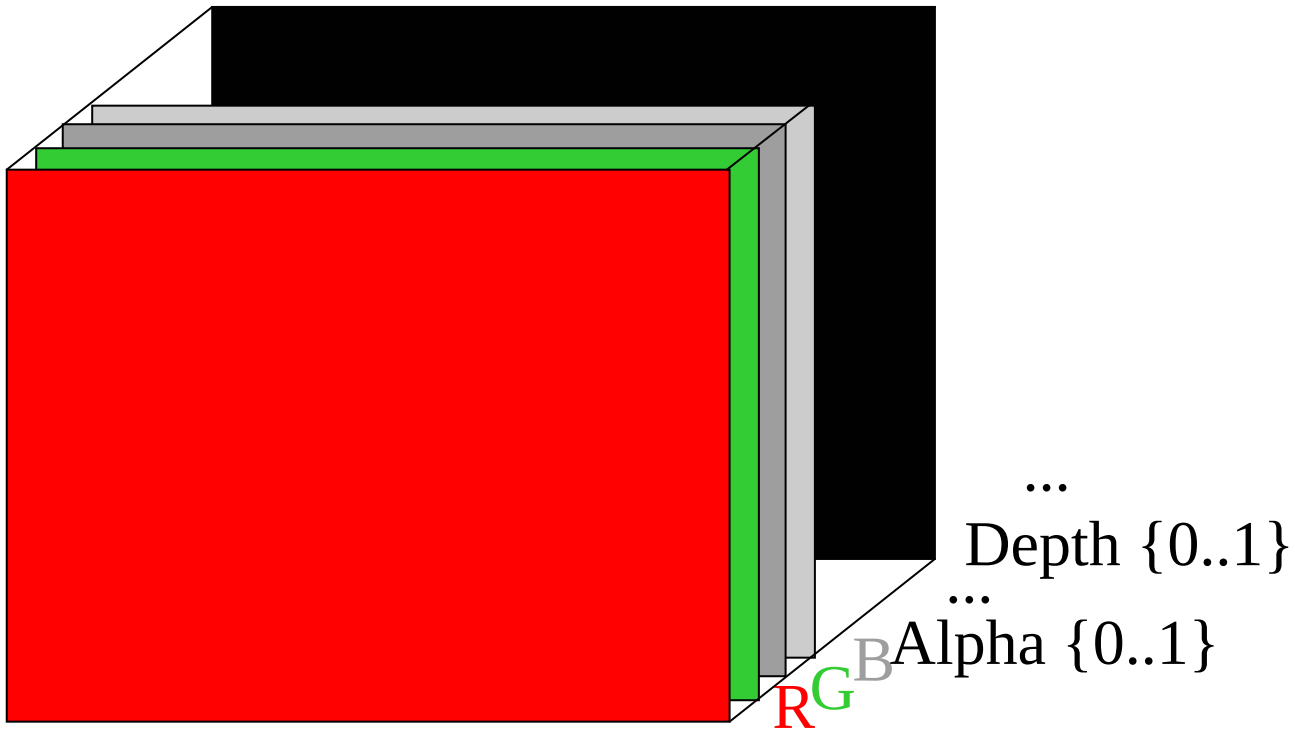
In Front In Back	Real Objects	Virtual Objects
Real Objects	optical occlusion	
Virtual Objects		



Occlusions between Real and Virtual Objects

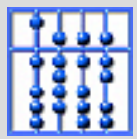
In Front In Back	Real Objects	Virtual Objects
Real Objects	optical occlusion	
Virtual Objects		graphics hardware (z-buffering)

- Framebuffer components



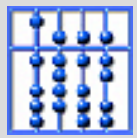
Depth Buffer Manipulation

- **glClearDepth (1.0);**
- **glClear (GL_DEPTH_BUFFER_BIT);**
- **glEnable (GL_DEPTH_TEST);**
- **glDepthFunc (GL_LEQUAL);**
- **glDepthMask (GL_TRUE);**



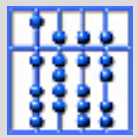
Occlusions between Real and Virtual Objects

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

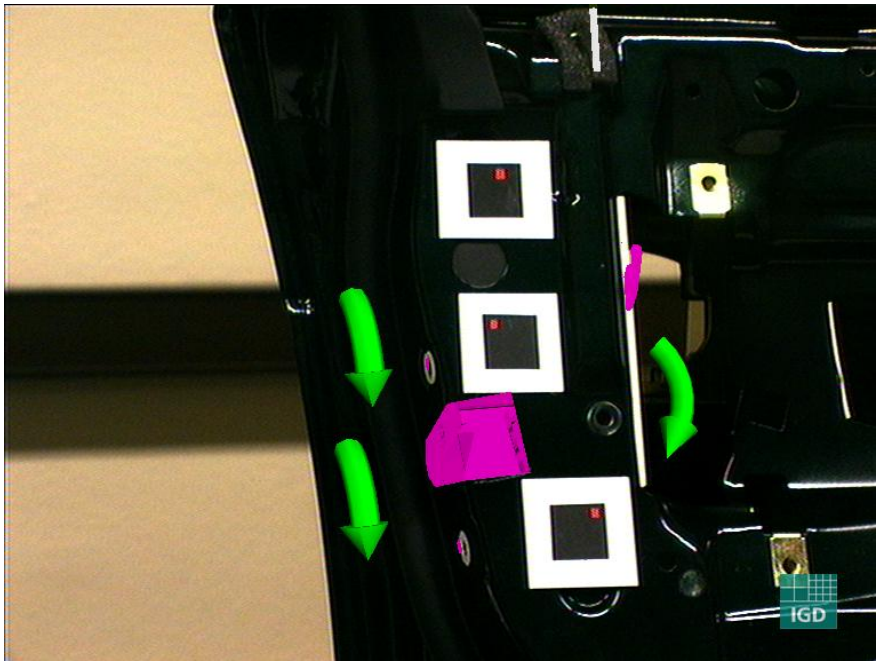
- **Orthographic view**
  glOrtho (0,imgwidth, 0,imgheight, -1.0,1.0);
- **Initialize modelling/viewing transformation**
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity();
- **Flip image (image array starts at upper left corner, OpenGL starts at lower left corner)**
  glPixelZoom(1.0, -1.0);
- **Define origin of image coordinate system:**
  glRasterPos2i (0, imgheight);

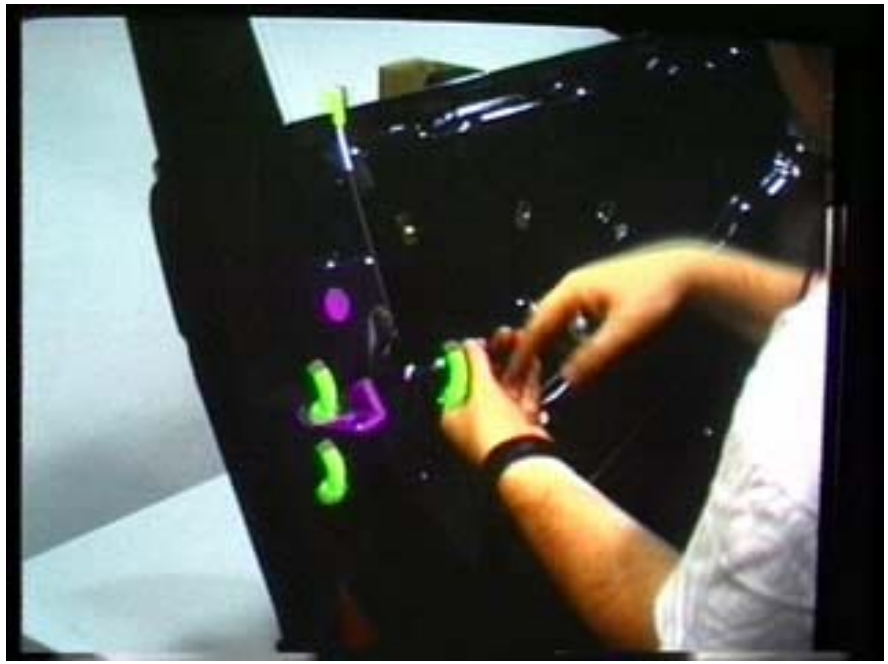


Drawing 2D images and marks

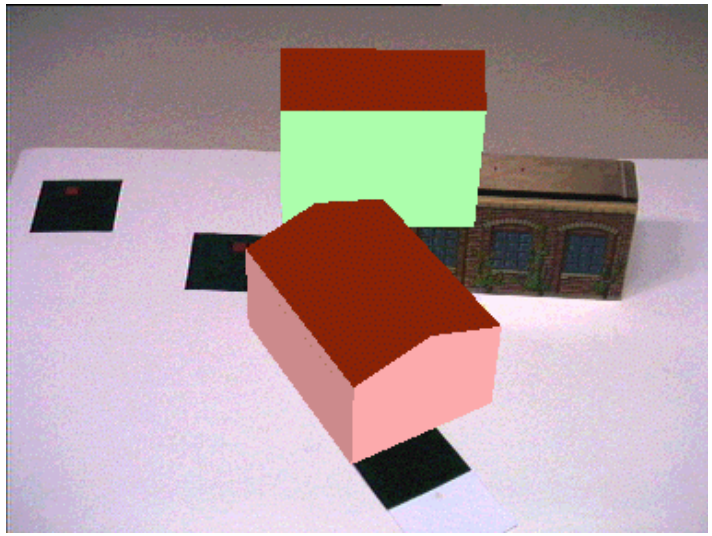
- **Draw image (upper left corner at raster pos)**
  glDrawPixels (imgwidth, imgheight, GL_RGBA, GL_UNSIGNED_BYTE, data);
- **Draw 2D marks (stored in a display-list) onto the image**
  glCallList (MyMarkers);

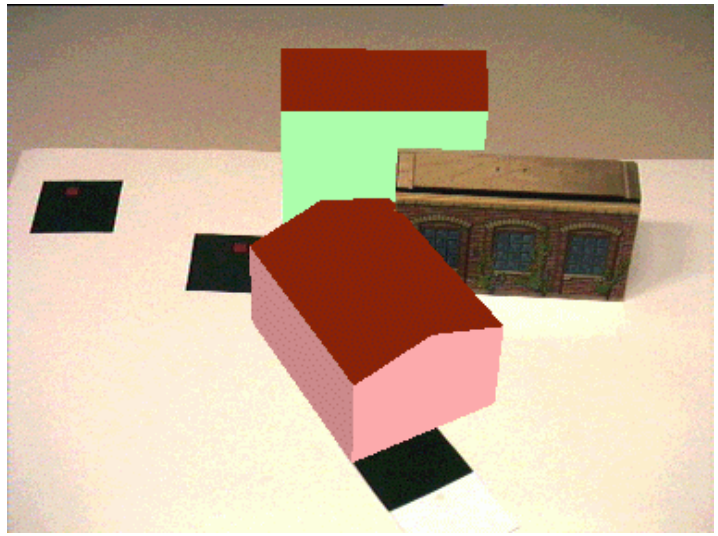
```
glNewList (MyMarkers, GL_COMPILE);
        glBegin (GL_LINES);
        glVertex2d (x, imgheight-y);
        ....
        glEnd ();
glEndList ();
```


IGD



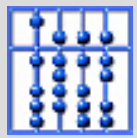
Integrating Virtual Objects into a Real Scene





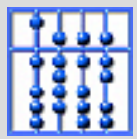
Occlusions between Real and Virtual Objects

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

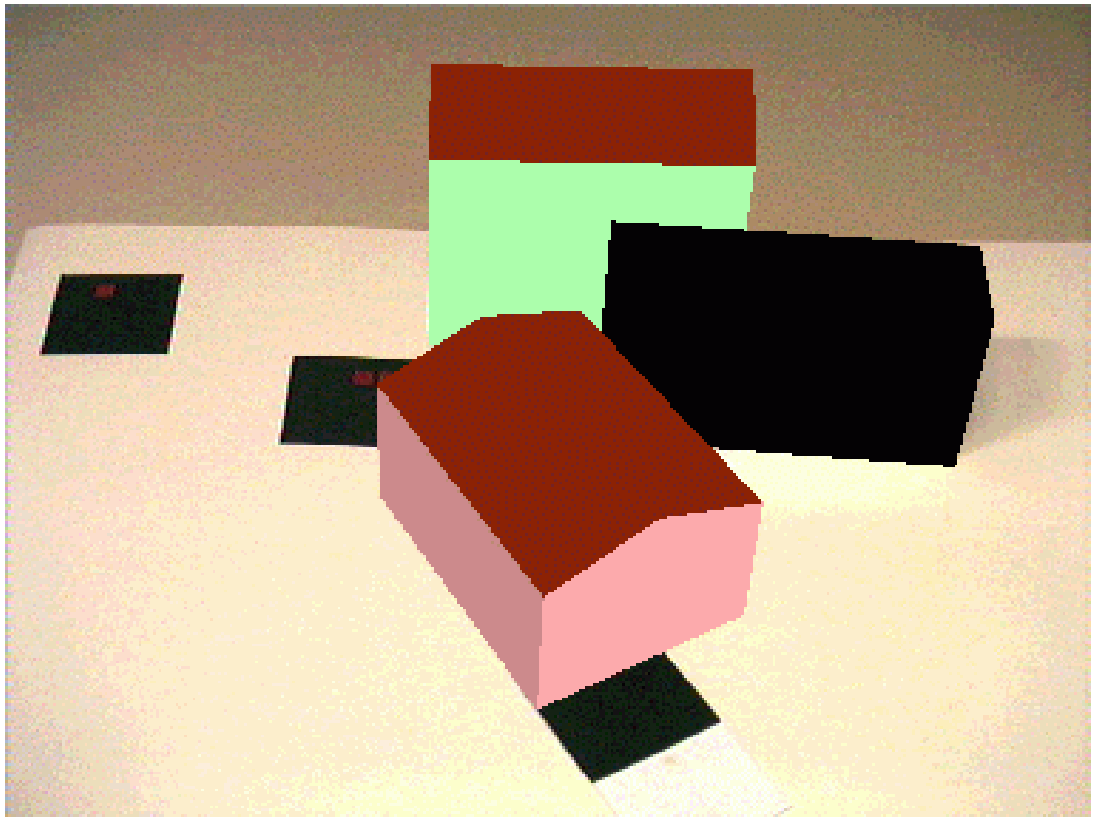


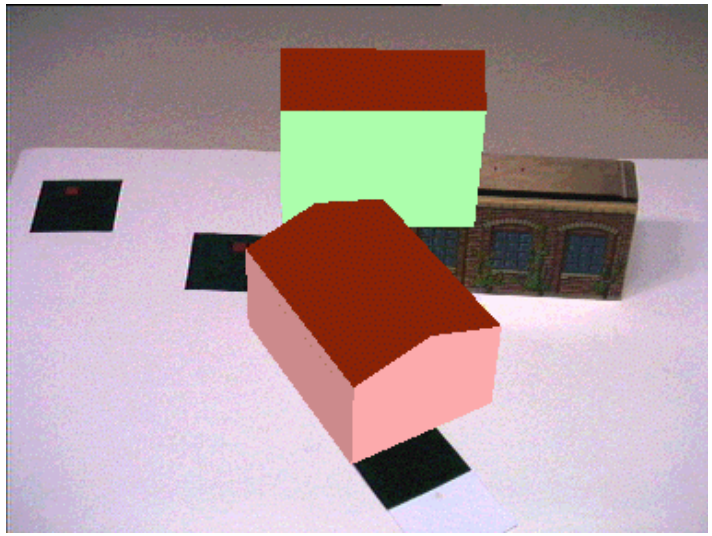
Integrating Virtual Objects into a Real Scene

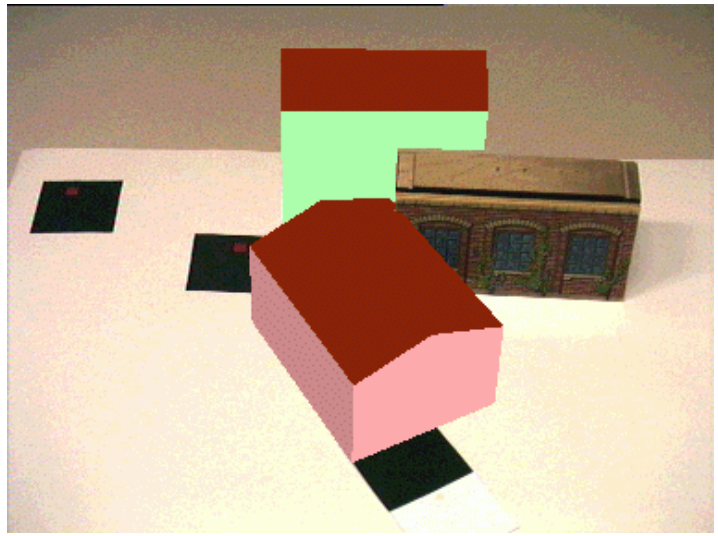
- Prequisites:
  - **Obtain geometrically precise models of all real objects.**
  - **Determine the current position of mobile objects.**
- For a given camera position:
  - **Draw video image (as background).**
  - **Initialize depth buffer.**
  - **Draw transparent models of real objects.**
  - **Draw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



Integrating Virtual Objects into a Real Scene

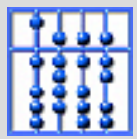


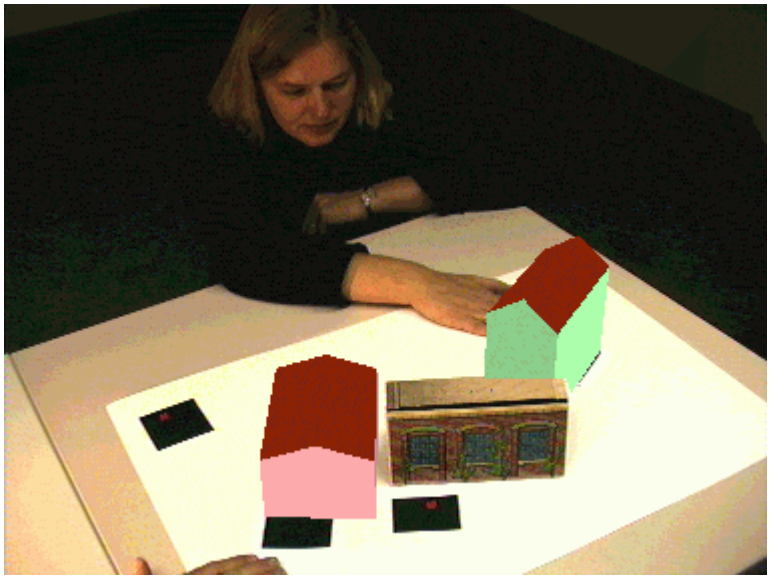


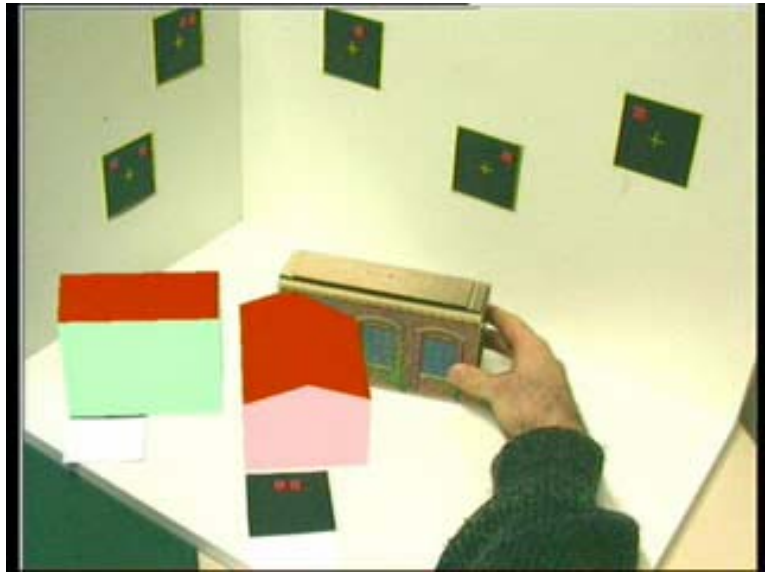


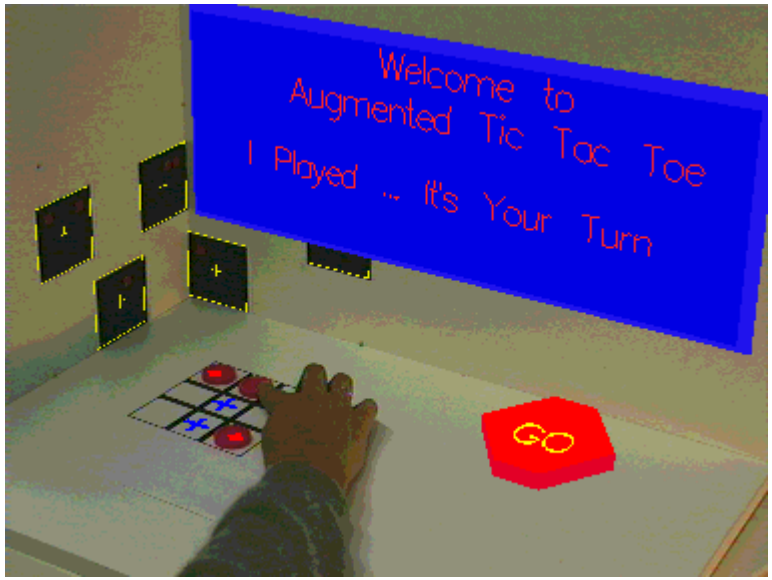
Drawing Transparent Pixels - Color Mask -

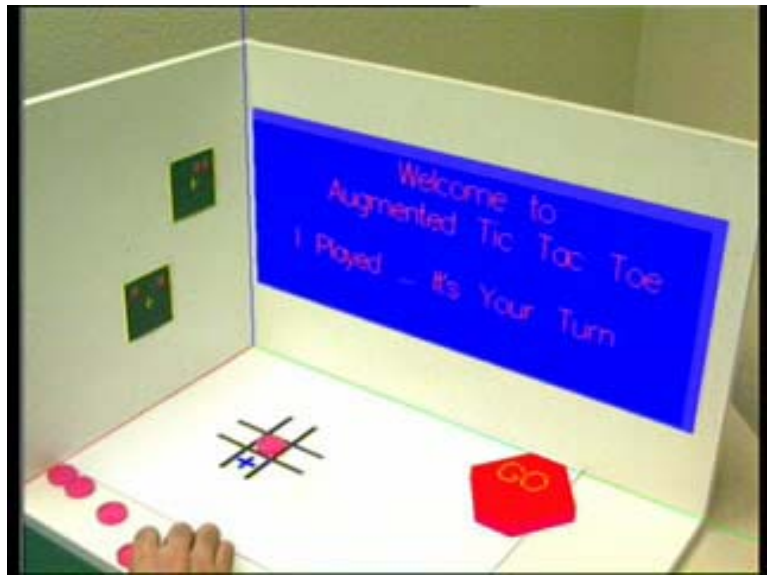
- **…Draw video image**
- **…Initialize depth buffer**
- **glColorMask (GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);**
- **…Draw reality model**
- **glColorMask (GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);**
- **…Draw virtual model**








Welcome to
Augmented Tic Tac Toe
I Played ... It's Your Turn
GO


Welcome to
Augmented Tic Tac Toe
I Played ... It's Your Turn
GO

Transparency via Pixel Blending

Updated pixel color [x,y] =

S * new color [x,y] *"source"*

+ D * current color [x,y] *"destination"*

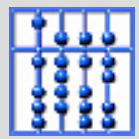
$$\begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix} = \begin{pmatrix} R_s\ S_r \\ G_s\ S_g \\ B_s\ S_b \\ A_s\ S_a \end{pmatrix} + \begin{pmatrix} R_d\ D_r \\ G_d\ D_g \\ B_d\ D_b \\ A_d\ D_a \end{pmatrix}$$

Source factor

Destination factor

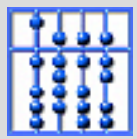
Blending Factors

- **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**