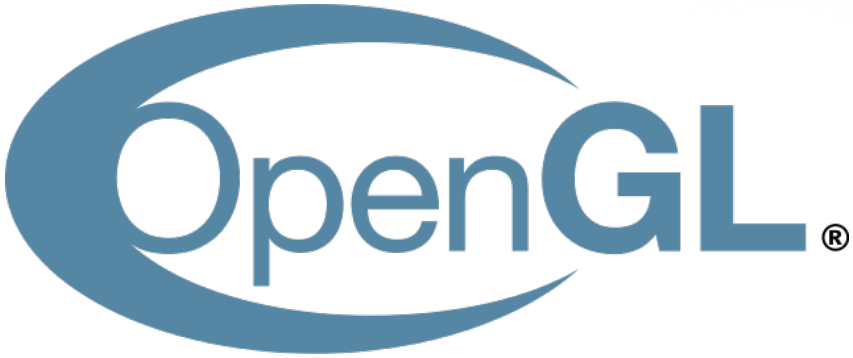


Thomas Höllt / Prof. Dr. Elmar Eisemann
Computer Graphics and Visualization
TU Delft

Graphics API

- Two major types: DirectX & OpenGL
  - Allows us to send commands to the card
  - Relatively close to hardware
- Recent push to decrease overhead
  - DX 12
  - Vulkan
  - Metal

Graphics API

- We will use OpenGL

- OpenGL is a state machine
  ➥ activate something and it stays activated

http://tinyurl.com/OpenGLTutorial

OpenGL Initialization

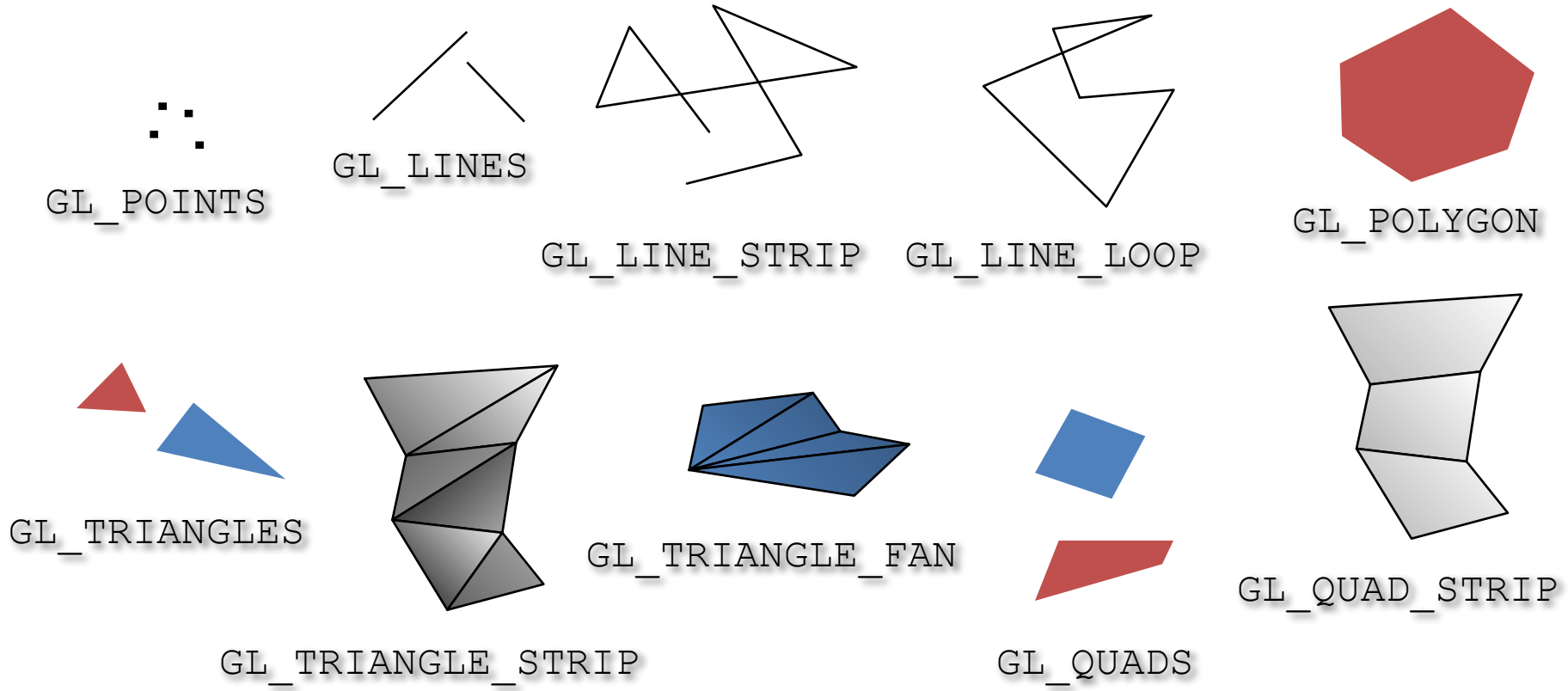
- Set up whatever state you're going to use

```
void init( void )
{
  glClearColor( 0.0, 0.0, 0.0, 1.0 );
  glClearDepth( 1.0 );

  glEnable( GL_LIGHT0 );
  glEnable( GL_LIGHTING );
  glEnable( GL_DEPTH_TEST );
}
```

OpenGL Geometric Primitives

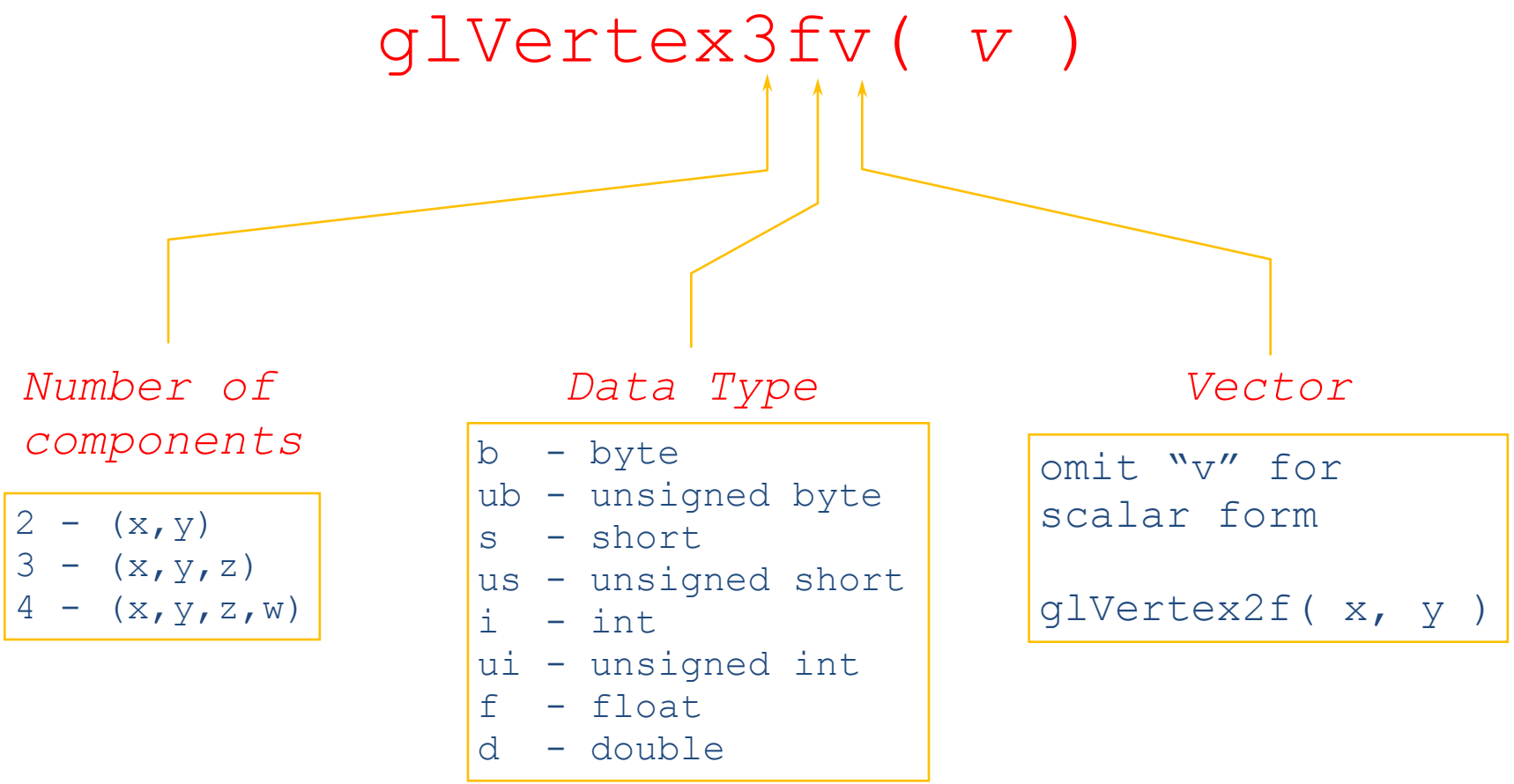
- All geometric primitives are specified by vertices



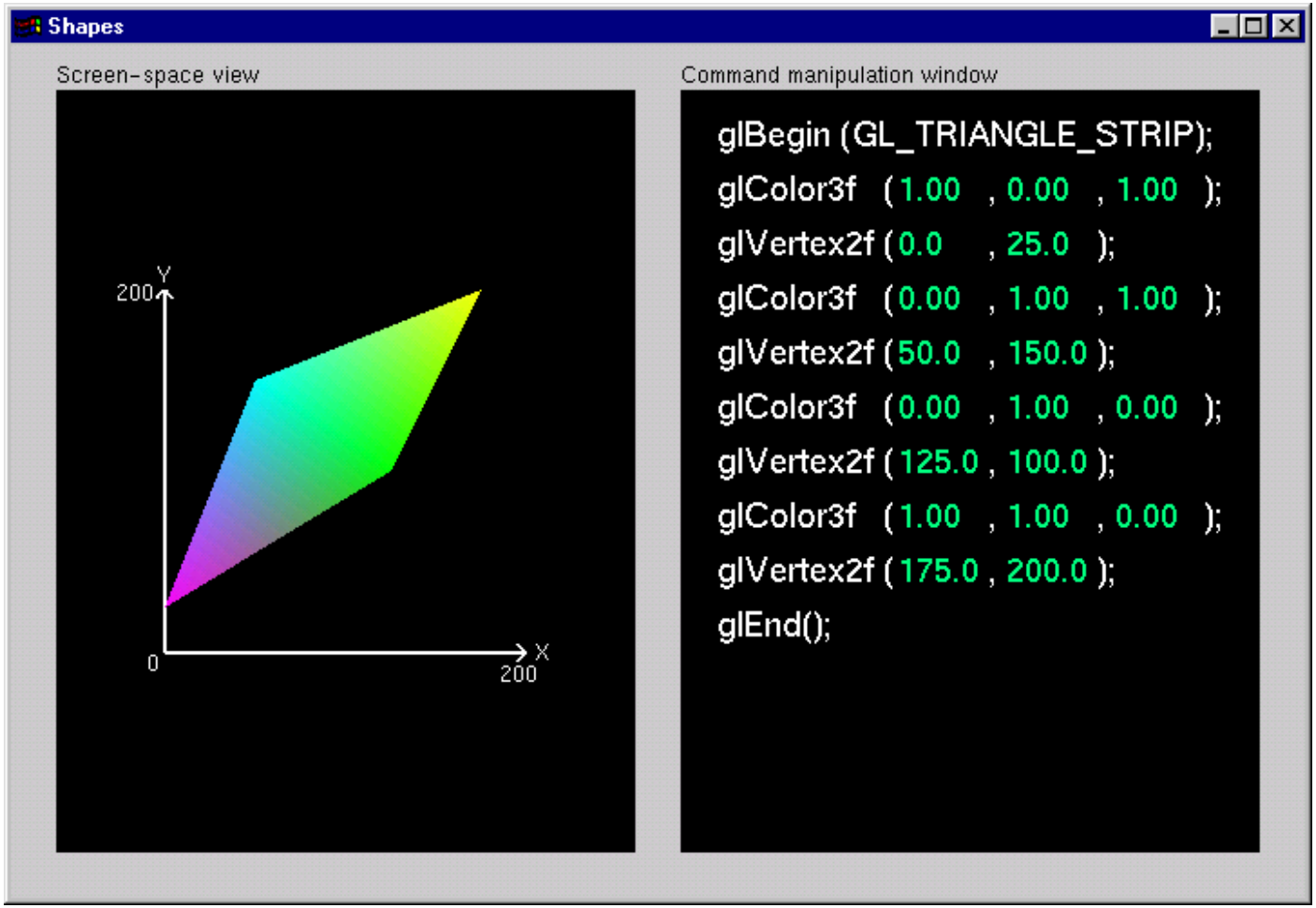
Simple Example

```
void drawRhombus( GLfloat color[] )
{
  glBegin( GL_QUADS );
  glColor3fv( color );
  glVertex2f( 0.0, 0.0 );
  glVertex2f( 1.0, 0.0 );
  glVertex2f( 1.5, 1.118 );
  glVertex2f( 0.5, 1.118 );
  glEnd();
}
```

OpenGL Command Formats



Shapes Tutorial



http://tinyurl.com/OpenGLTutorial

Cameras and Projection

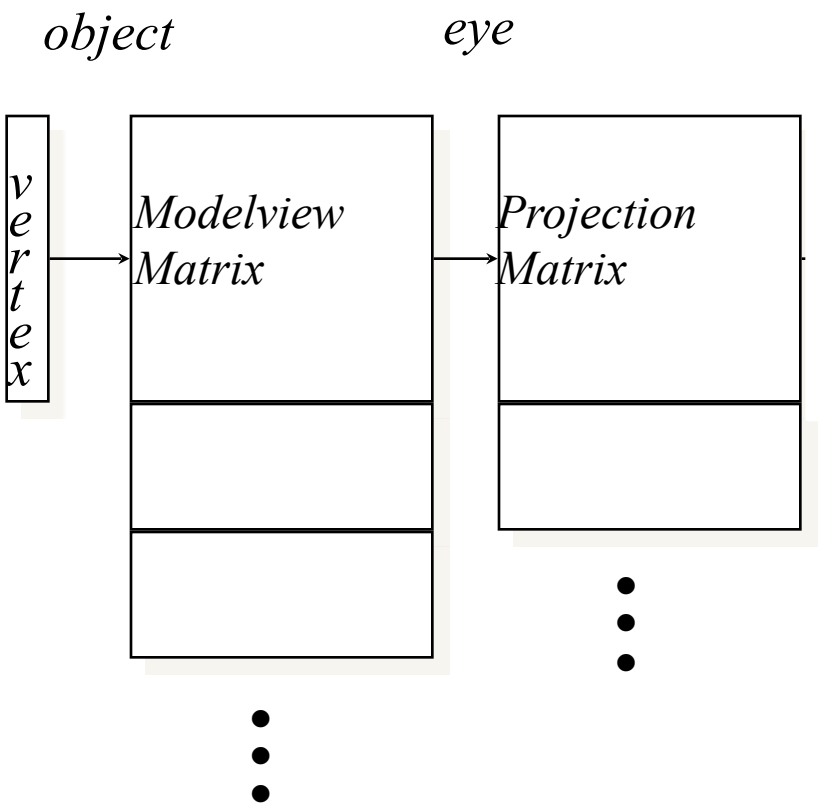
Transformations in OpenGL

- Modeling
- Viewing
  - orient camera
  - projection
- Animation
- Map to screen

3D Transformations

- A vertex is transformed by 4 x 4 matrices
  - all affine operations are matrix multiplications
  - all matrices are stored column-major in OpenGL
  - matrices are always post-multiplied
  - product of matrix and vector is a vector

Transformation Pipeline



Matrix Operations

- Viewport (pixel mapping – in all exercises, no need to touch)

  glViewport(*x, y, width, height*)
  = where to draw on the screen

- Transformations
- Specify Current Matrix Stack

  glMatrixMode(*GL_MODELVIEW*)
  Or glMatrixMode(*GL_PROJECTION*)

- Other Matrix or Stack Operations

  glLoadIdentity() glLoadMatrix glMultMatrix
  glPushMatrix() glPopMatrix()
  – Use friendly calls of glMultMatrix:
  glRotate glTranslate glScale

Specifying Transformations

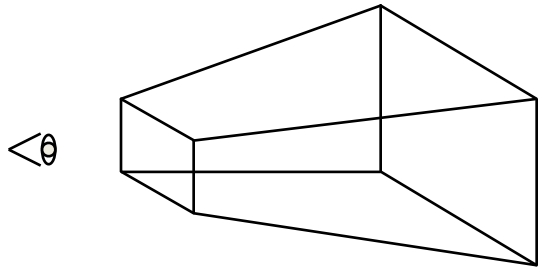
- Two styles of specifying transformations
  - specify matrices (**`glLoadMatrix, glMultMatrix`**)
  - specify operation (**`glRotate, glOrtho`**)

- Programmers do not have to remember exact matrices
  - check appendix of Red Book (Programming Guide)

Programming Transformations

- Prior to rendering, view, locate, and orient:
  - eye/camera position
  - 3D geometry
- Manage the matrices
  - including matrix stack
- Combine (composite) transformations

Projection Transformation



- Shape of viewing frustum
- Perspective projection

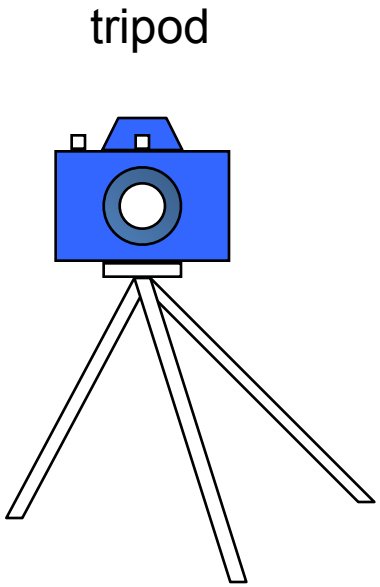
```
gluPerspective( fovy, aspect, zNear, zFar )
```

- Orthographic parallel projection

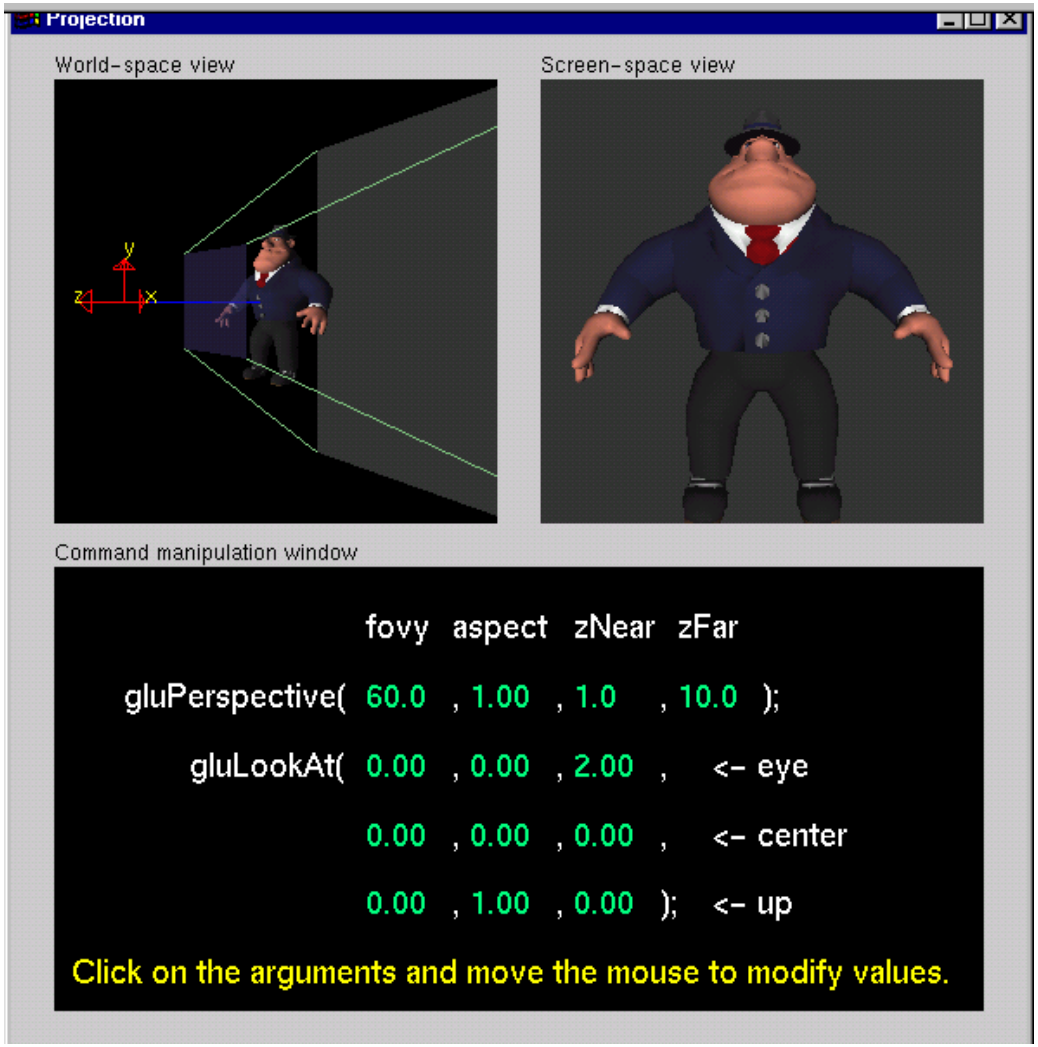
```
glOrtho( left, right, bottom, top, zNear, zFar )
```

Viewing Transformations

- Position the camera/eye in the scene
  - place the tripod down; aim camera
- To “fly through” a scene
  - change viewing transformation and redraw scene
- `gluLookAt( eye`$_x$`, eye`$_y$`, eye`$_z$`,`
  `aim`$_x$`, aim`$_y$`, aim`$_z$`,`
  `up`$_x$`, up`$_y$`, up`$_z$ `)`
  - up vector determines unique orientation
  - careful of degenerate positions



Projection Tutorial



http://tinyurl.com/OpenGLTutorial

Modeling Transformations

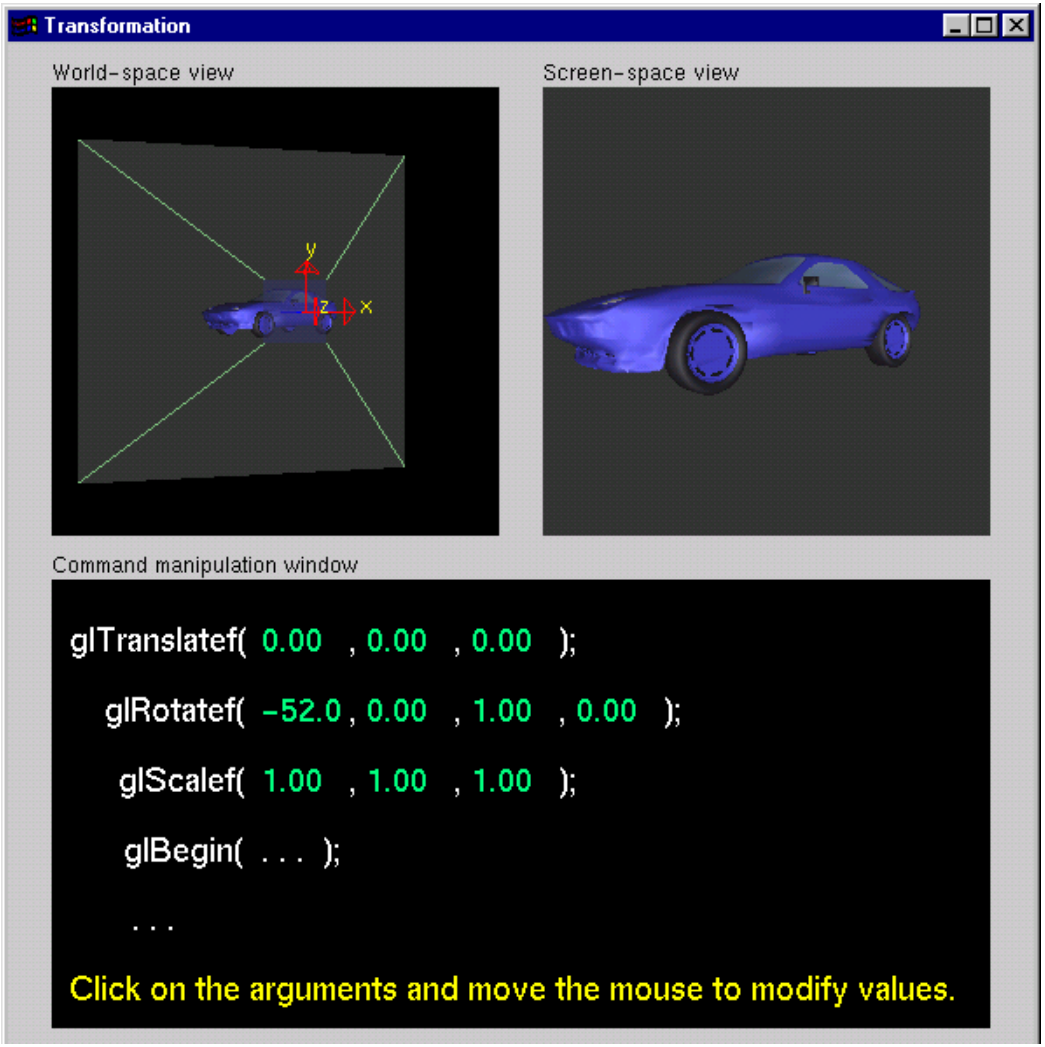
- Move object

  `glTranslate{fd}( x, y, z )`
- Rotate object around arbitrary axis

  `glRotate{fd}( angle, x, y, z )`
  - angle is in degrees
- Dilate (stretch or shrink) or mirror object

  `glScale{fd}( x, y, z )`

Transformation Tutorial



http://tinyurl.com/OpenGLTutorial

Object Hierarchies

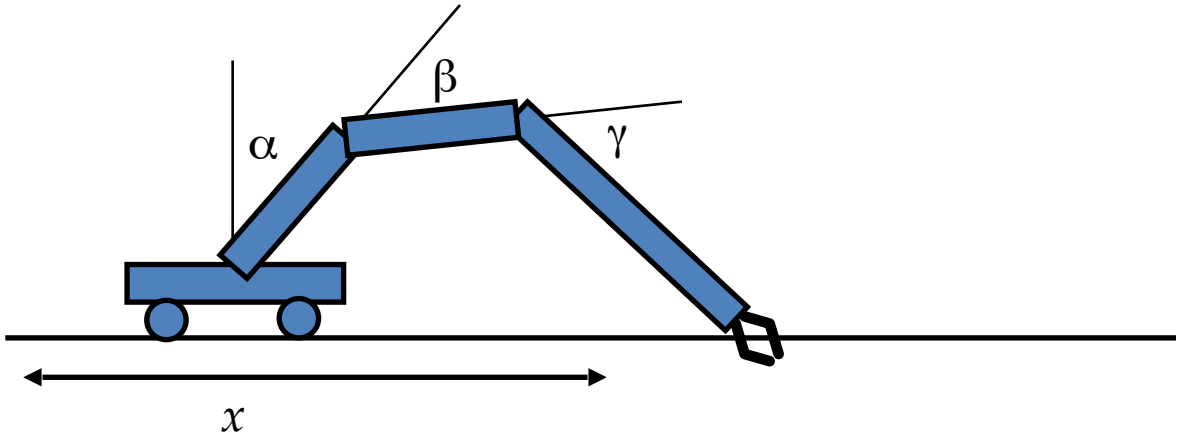
MyFunctionToDrawObjects

- Concatenate operations

- Use relative coordinates:
  - Position of hand with respect to arm
  - Position of arm with respect to body
  - …

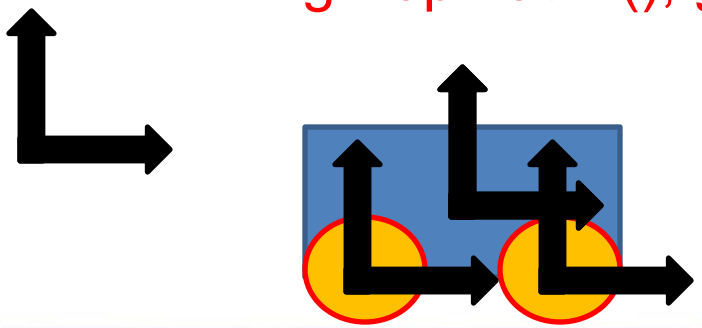
Why concatenate operations?

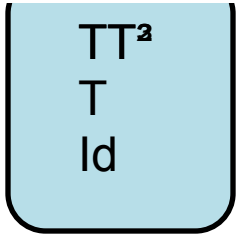
- Often, objects are defined as combinations
  - Examples : robots, cars (wheels)…
- We want a natural behavior:
  - Object stays connected:
    - If you move the arm, the hand should follow



Matrix Stack

- Keep information about modifications
  - glPushMatrix(); **T** (glTranslatef(x,y,z); to position)
  - draw car body
  - glPushMatrix(); **TT^2** (glTranslatef(u,v,w); to 1st wheel)
  - draw first wheel as circle of center (0,0)
  - return to **T:** glPopMatrix();
  - glPushMatrix(); **TT^3** (glTranslatef(r,s,t); to 2nd wheel)
  - draw second wheel
  - glPopMatrix(); glPopMatrix();





Manipulating OpenGL State

- Appearance is controlled by current state

```
for each ( primitive to render ) {
    update OpenGL state
    render primitive
}
```

- Manipulating vertex attributes is most common way to manipulate state

```
glColor*()
glNormal*()
glTexCoord*()
```

Controlling current state

- Setting State, e.g.,

```
glPointSize( size );
glShadeModel( GL_SMOOTH );
```

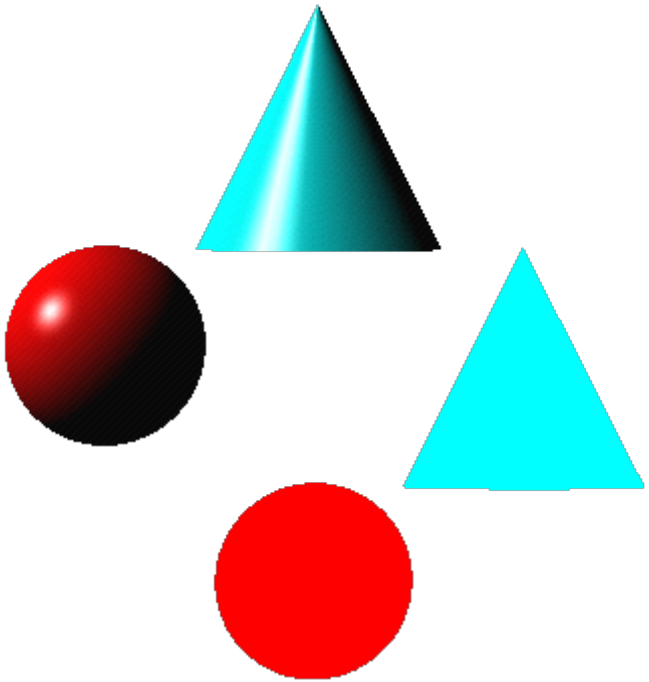
- Enabling Features, e.g.,

```
glEnable( GL_LIGHTING );
```

Lighting

Lighting Principles

- Lighting simulates how objects reflect light
  - material composition of object
  - light's color and position
  - global lighting parameters
    - ambient light
    - two sided lighting



How OpenGL Simulates Lights

- Phong lighting model
  - Computed at vertices
- Lighting contributors
  - Ambient (global)
  - Diffuse (material, light)
  - Specular (material, light)
  - Emission (material)

Surface Normals

- Normals define how a surface reflects light

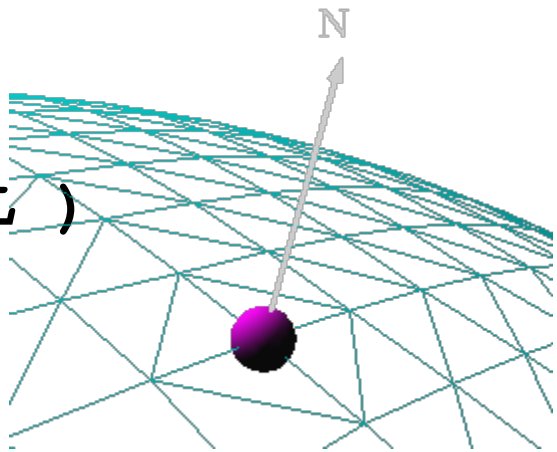
```
glNormal3f( x, y, z )
```

  - Current normal is used to compute vertex's color
  - Use *unit* normals for proper lighting
    - scaling affects a normal's length

`glEnable( GL_NORMALIZE )`

or

`glEnable( GL_RESCALE_NORMAL )`



Material Properties

- Define the surface properties of a primitive

```
glMaterialfv( face, property, value );
```

  – separate materials for front and back

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

Ambient Term

- Very simplistic
  - No real physical basis…
  - No indications on the shape of an objet!

K_a croissant

- But often used in practice
  - Emulates indirect light from scene
  - Elements in shadow are not completely black

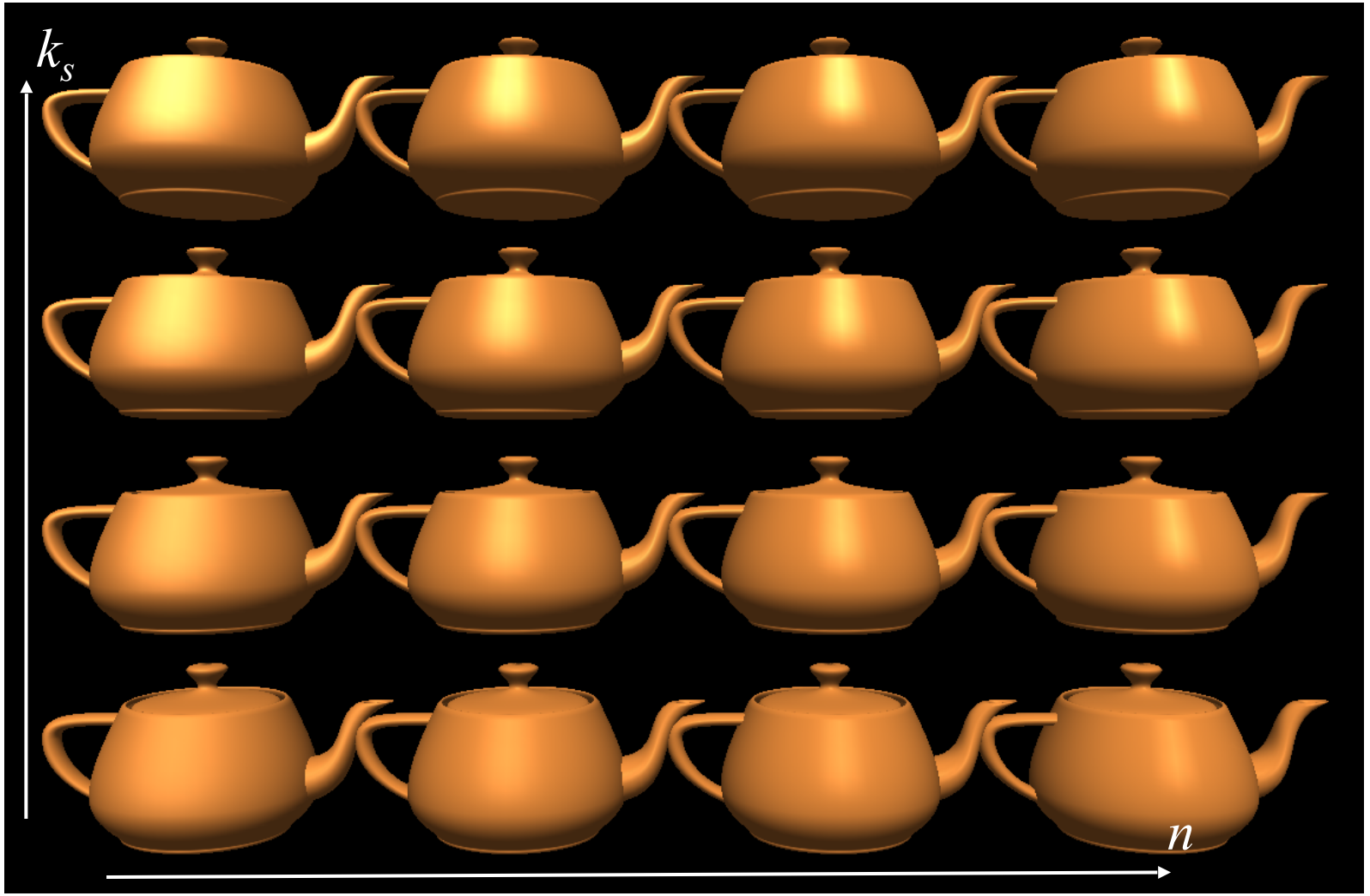
Diffuse Term

- Hypothesis: isotropic *a.k.a* lambertian
- Shading varies along surface
  - Gives information about shape

K_d increasing

  - Does not depend on observer position!

Specular Term



Light Properties

```
glLightfv( light, property, value );
```

- ***light*** specifies which light
  - multiple lights, starting with **GL_LIGHT0**

    glGetIntegerv(*GL_MAX_LIGHTS*, *&n*);
- ***properties***
  - **colors** (GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR)
  - **position** (GL_POSITION)
  - **attenuation** (GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION)

Types of Lights

- OpenGL supports two types of Lights
  - Local (Point) light sources
  - Infinite (Directional) light sources
- Type of light controlled by w coordinate
  - Remember points at infinity! w=0 -> directional

Turning on the Lights

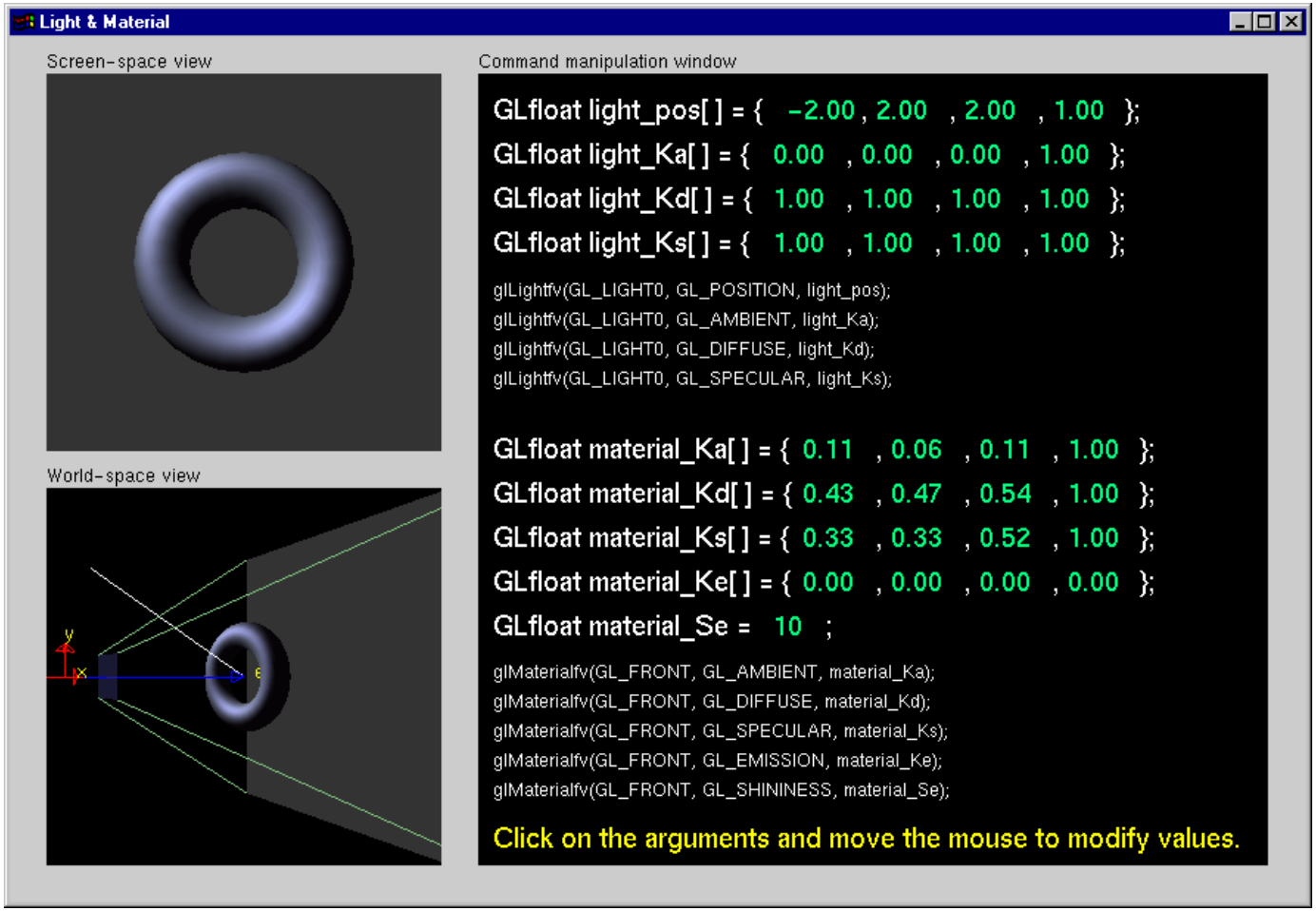
- Flip each light's switch

```
glEnable( GL_LIGHTn );
```

- Turn on the power

```
glEnable( GL_LIGHTING );
```

Light Material Tutorial



http://tinyurl.com/OpenGLTutorial

What we use: in practice!

- OpenGL mode: glEnable(GL_COLOR_MATERIAL);

  makes color become a material

  Hence: glColor defines a diffuse material

  Then you only need to define, LightPos:

```
glLightfv( GL_LIGHT0, GL_POSITION, pos );
```

That is all for today…

Books

- OpenGL Programming Guide
- OpenGL Reference Manual
- OpenGL Programming for the X Window System
  - includes many GLUT examples
- Interactive Computer Graphics: A top-down approach with OpenGL

Thank you very much
for your attention!

…and thanks to M. Kilgard for help with slides…