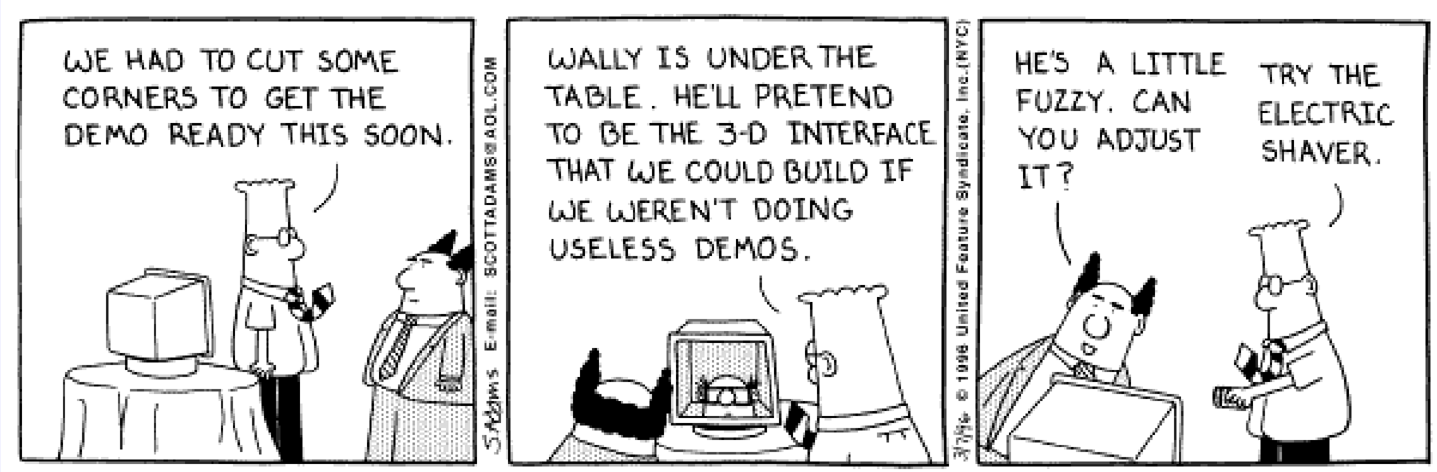
SommerCampus 2004

An Introduction to Computer Graphics using OpenGL

Marco Reisert and Lokesh Setia
{reisert, setia} @informatik.uni-freiburg.de



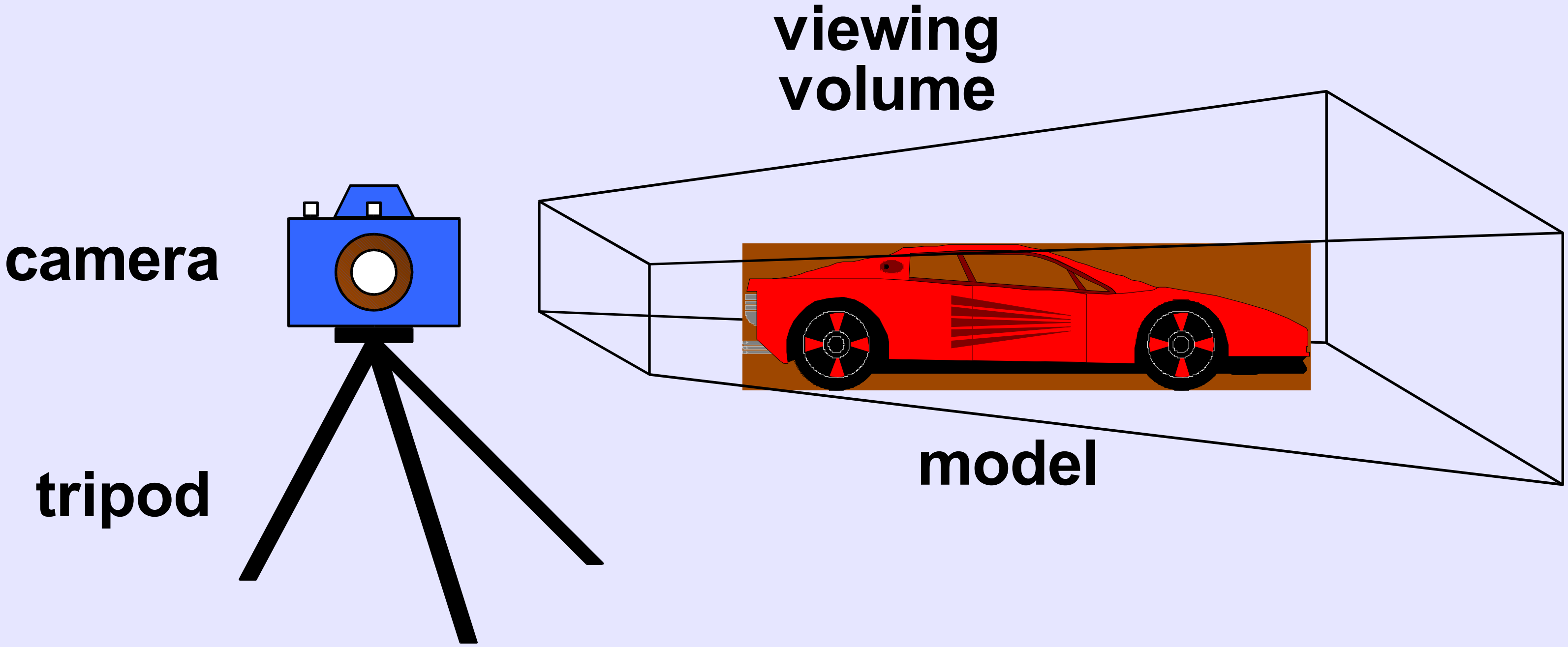
Copyright © 1996 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited

What is OpenGL?

- OpenGL is a software interface for 3-D computer graphics
- Originally developed by SGI (IRIS GL); since 1992 under control of ARB (**A**rchitecture **R**eview **B**oard)
- OpenGL specification is hardware-independent, window-system independent and operating system independent.
- Different implementations (as Hardware and/or Software) are possible!

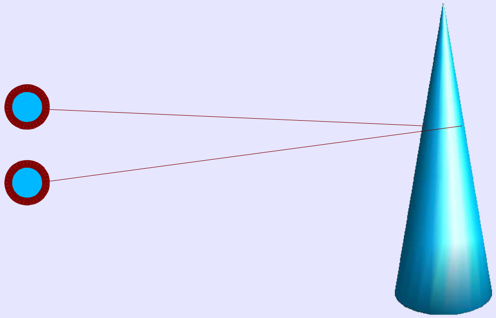
Computer Graphics using OpenGL

3D computer graphics is just like taking photographs in a virtual mathematical world!



3D and OpenGL

- Human Beings perceive depth primarily by stereo vision, i.e. slight differences in two eye images enable our brain to estimate depths of objects



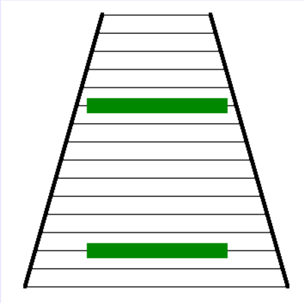
- Images rendered by OpenGL on a computer monitor **cannot** achieve this effect by itself.

- However using accessories like **video glasses** or shutter-glasses one can achieve „realistic“ Virtual Reality

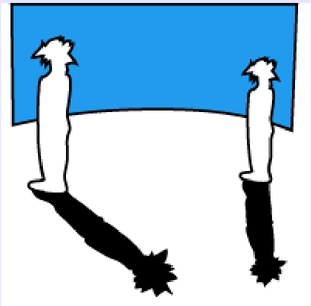
* Garry Kasporav playing with Virtual Chess Pieces on Nov. 11 2003

3D Graphics and OpenGL

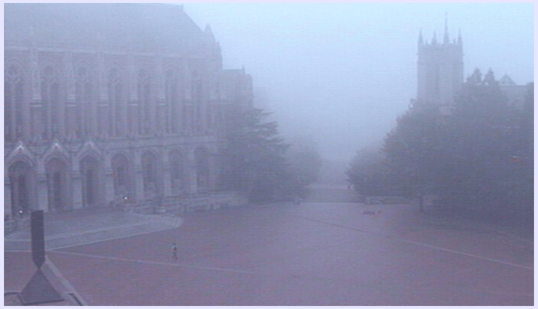
There are however other ways through which we get a feel for depth and realism



Perspective cues
Size of objects decreases as they move away from us



Shadows
3D objects cast shadows depending on their shape



Fog / Atmospheric effects
Objects near to us are clearer than those far away

With OpenGL, one can create all these effects !

Homogeneous Coordinates

- A 4-valued homogeneous coordinate **v** = $(x, y, z, w)^T$ corresponds to the 3-D coordinate **v** = $(x/w, y/w, z/w)^T$

- OpenGL internally stores all vertices as homogeneous coordinates.

- The benefit of homogeneous coordinates is that most common transformations (translation, rotation, scaling, perspective projection etc.) can simply be represented by multiplication with a matrix **M**. i.e. **v' = M.v**

- Examples of homogeneous coordinates:

$$(2, 4, 0, 1)^T \equiv (2, 4, 0)^T$$
$$(2, 4, 0, 2)^T \equiv (1, 2, 0)^T$$
$$(1, 2, 0, 1)^T \equiv (1, 2, 0)^T$$

$$\begin{bmatrix} \overset{\text{Rotation, Scaling, ...}}{\overset{M}{\boxed{\begin{matrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ m31 & m32 & m33 \end{matrix}}}} & \overset{\text{Translation}}{\overset{T}{\boxed{\begin{matrix} Tx \\ Ty \\ Tz \end{matrix}}}} \\ \begin{matrix} 0 & \quad 0 & \quad 0 \end{matrix} & 1 \end{bmatrix} * \begin{bmatrix} \overset{\text{Vertex}}{\overset{v}{\boxed{\begin{matrix} Vx \\ Vy \\ Vz \end{matrix}}}} \\ 1 \end{bmatrix} = \begin{bmatrix} \boxed{M*v + T} \\ 1 \end{bmatrix}$$

$$(1, 2, 0, 0)^T \rightarrow \text{Point at Infinity!}$$

General steps in using OpenGL

1. Initialize a window to draw into (not part of OpenGL package).
2. Place objects in a 3-D mathematical world
3. Select light sources
4. Select a point to look from, and the viewing frustum
5. Instruct OpenGL to draw!

Related APIs

- **AGL, GLX, WGL**
  - glue between OpenGL and windowing systems
- **GLU (OpenGL Utility Library)**
  - part of OpenGL
  - contains utility functions such as setting viewing orientations, Polygon tessellators etc.
- **GLUT (OpenGL Utility Toolkit)**
  - portable windowing API
  - not officially part of OpenGL

- We would use the **Qt** library for windowing related tasks.

OpenGL and Qt

The graphics library Qt provides support for OpenGL using four classes: **QGLWidget**, **QGLContext**, **QGLFormat** and **QGLColormap**.

Most applications need only **QGLWidget** class. Important member functions to be implemented are:

- **initializeGL**(): called once before other functions. Useful for initializing OpenGL parameters.
- **resizeGL**(int width, int height): called when the size of the OpenGL window is changed. Useful for changing the camera frustum accordingly.
- **paintGL**(): OpenGL Drawing commands should be placed here

OpenGL as a Renderer

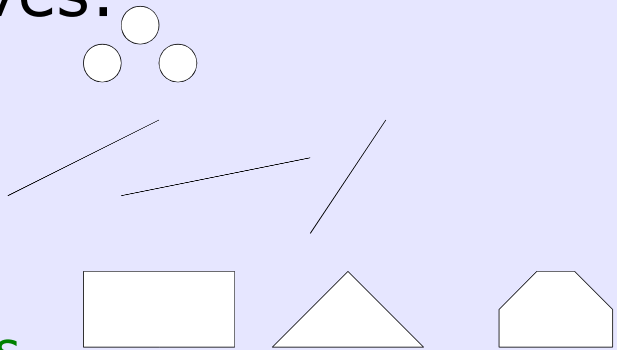
- **Geometric primitives**
  - points, lines and polygons
- **Image Primitives**
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on **state**
  - colors, materials, light sources, etc.

OpenGL as a State Machine

- OpenGL can be set into different modes (or states)
- They remain in effect until changed again (e.g. current pen color)
- Most state variables are simply boolean which can be set using glEnable() or glDisable()
- Current values can be queried using glIsEnabled(), or using glGetFloatv(), glGetIntegerv(), etc.
- Some state variables have more specific query command, e.g. glGetLight*();

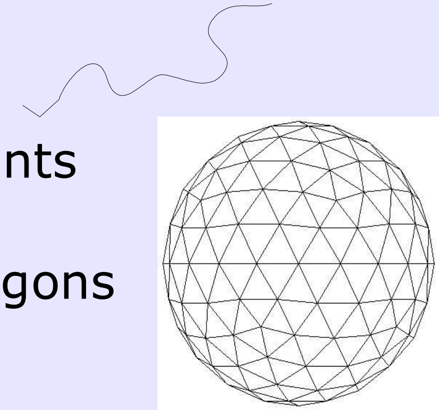
Geometric primitives

→ OpenGL supports only basic geometric primitives:



- Points
- Lines
- Polygons

→ Complex objects can however always be built using the basic primitives.

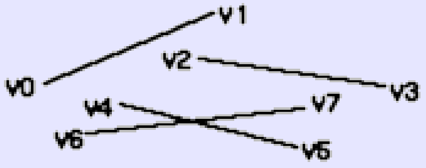


- Curves → using many small line segments
- Solid Objects → using many small polygons

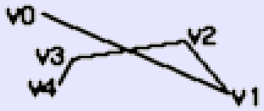
OpenGL Geometric Primitives

GL_POINTS

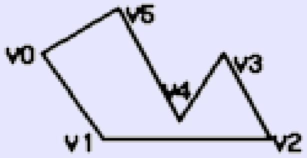
- **All geometric primitives are specified by vertices**



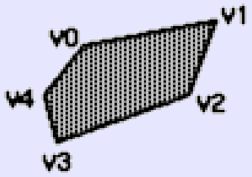
GL_LINES



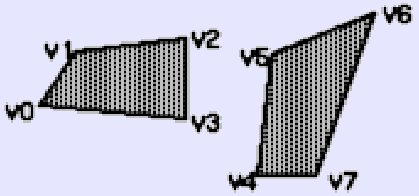
GL_LINE_STRIP



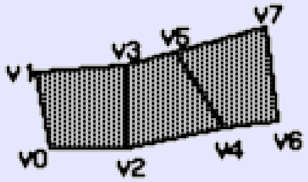
GL_LINE_LOOP



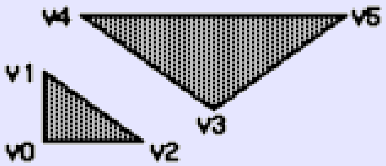
GL_POLYGON



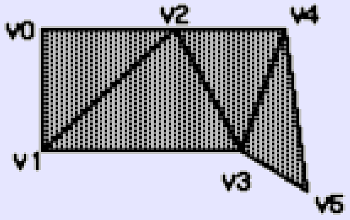
GL_QUADS



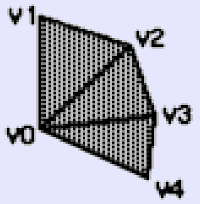
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_STRIP

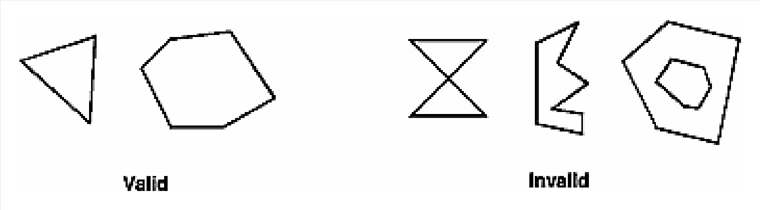


GL_TRIANGLE_FAN

Polygon Primitives

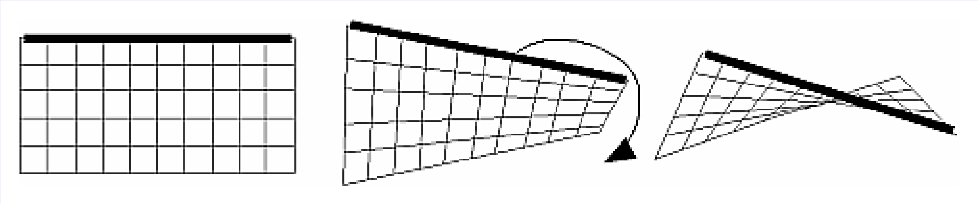
Valid and Invalid Polygons

→ Use Tessellation

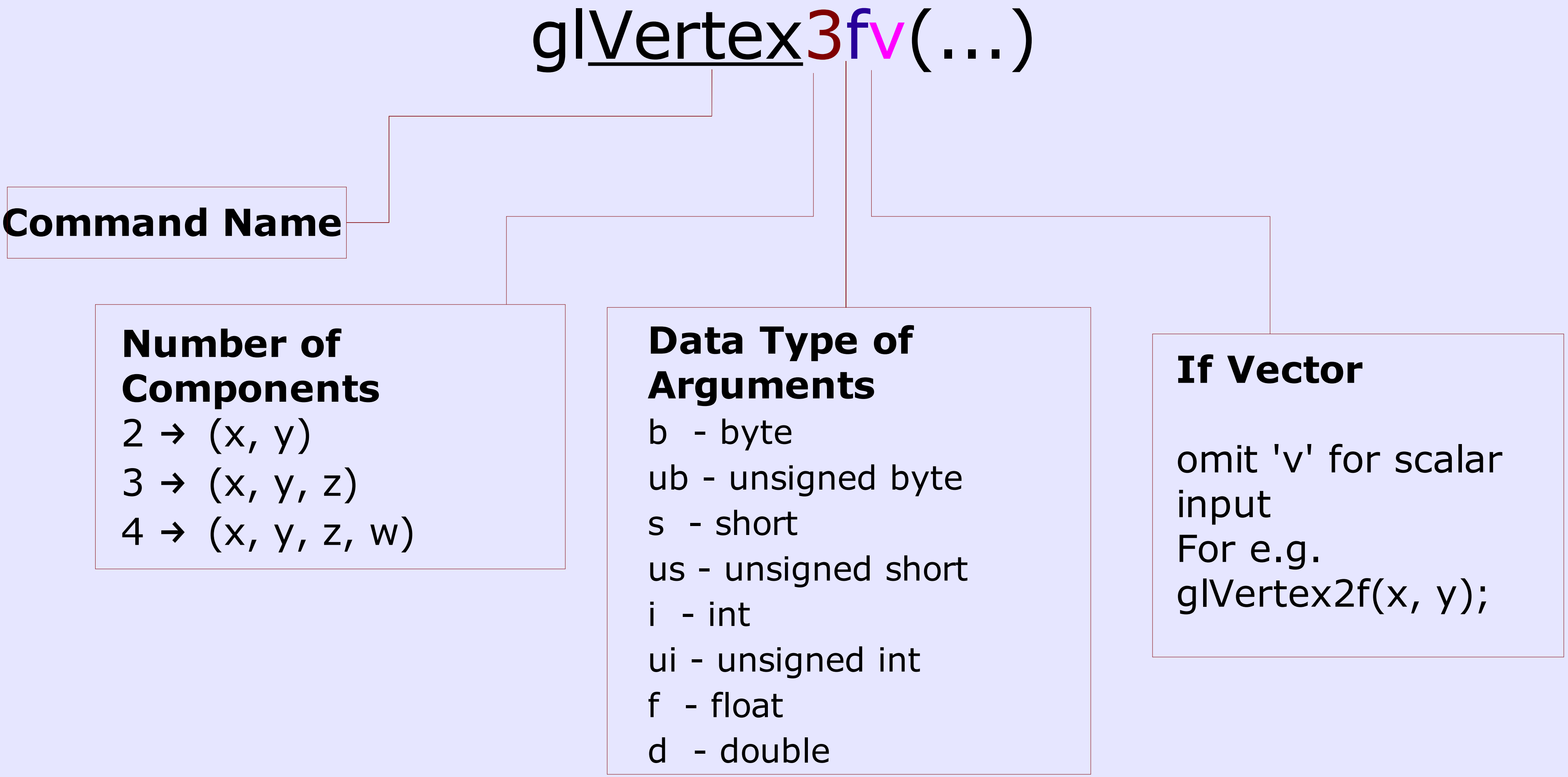


Planar and Non-Planar Polygons

→ Use triangles if possible



Function Naming Convention in OpenGL



glVertex3fv(...)

Command Name

Number of Components
2 → (x, y)
3 → (x, y, z)
4 → (x, y, z, w)

Data Type of Arguments

b	byte
ub	unsigned byte
s	short
us	unsigned short
i	int
ui	unsigned int
f	float
d	double

If Vector

omit 'v' for scalar input
For e.g.
glVertex2f(x, y);

Setting Colors

- Defining a color to be used for future drawing commands:

```
void glColor3{bdfi...} (TYPE red,  TYPE green, TYPE blue);
void glColor4{bdfi...} (TYPE red,  TYPE green, TYPE blue, TYPE alpha);
void glColor{34}{bdfi...}v (const TYPE* v);
```

Examples: // Set current color to yellow; all functions below give the same result

```
glColor3f(1.0, 1.0, 0.0);
glColor4f(1.0, 1.0, 0.0, 1.0);
glColor3ub(255, 255, 0);
GLdouble Yellow[] = {1.0, 1.0, 0.0, 1.0};  glColor4dv(Yellow);
```

- Background can be cleared using

```
void glClearColor   (GLclampf red, GLclampf green, GLclampf blue,
                     GLclampf alpha );
void glClear (GL_COLOR_BUFFER_BIT);
```

Using Geometric Primitives

- All Geometric Primitives must be enclosed by calls to glBegin(*prim_type*) and glEnd()

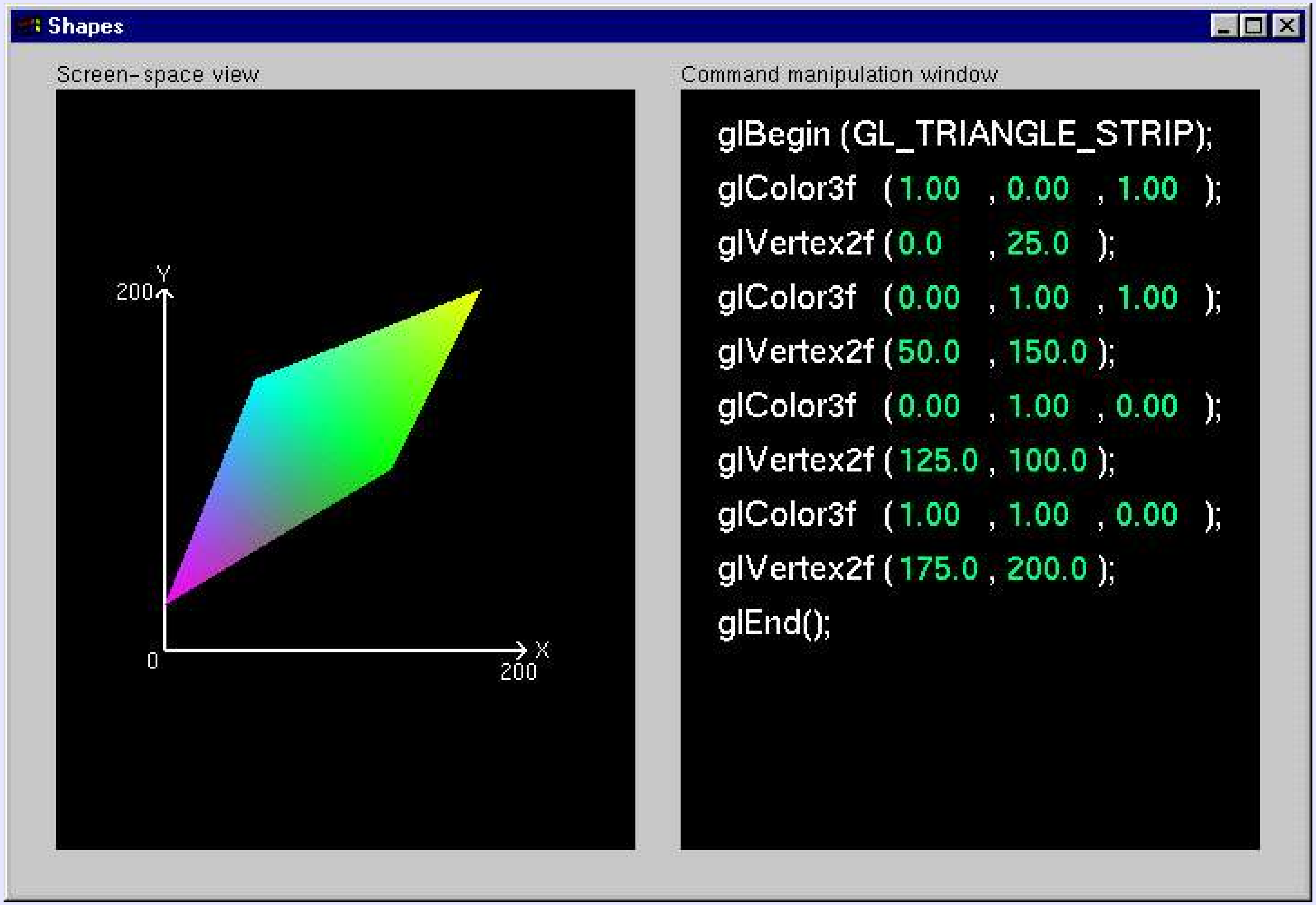
Setting State:

```
glPointSize( size );
glLineStipple( repeat,
  pattern );
```

Parameters can be specified either just once or per Vertex

```
void drawCircle (float x, float y, float
rad, int steps)
  {
  glLineWidth (1);
  glColor3f (0.0, 0.0, 0.0);   // black
  glBegin(GL_LINE_LOOP);

  for (int i=0; i<steps; ++i)
  {
    float angle = 2*M_PI*float(i)/float
 (steps);
    glVertex2f(
        x + rad*cos(angle),
        y + rad*sin(angle)
    );
  }
  glEnd();
 }
```

Example: setting parameters per Vertex



```
glBegin (GL_TRIANGLE_STRIP);
glColor3f  (1.00  , 0.00  , 1.00  );
glVertex2f (0.0    , 25.0  );
glColor3f  (0.00  , 1.00  , 1.00  );
glVertex2f (50.0   , 150.0 );
glColor3f  (0.00  , 1.00  , 0.00  );
glVertex2f (125.0 , 100.0 );
glColor3f  (1.00  , 1.00  , 0.00  );
glVertex2f (175.0 , 200.0 );
glEnd();
```

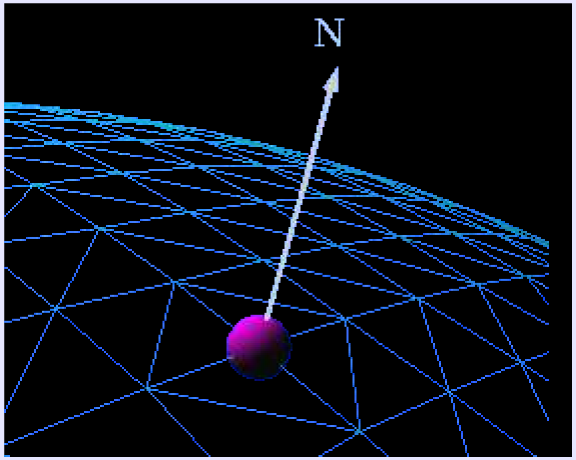
Surface Normals

- 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

  use glEnable(*GL_NORMALIZE*)

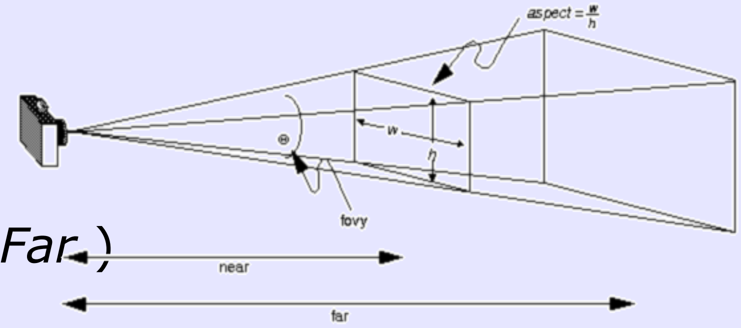


Viewing Transformations

- Perspective projection

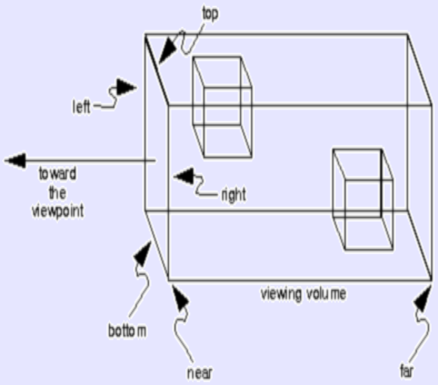
  gluPerspective(*fovy, aspect, zNear, zFar*)

  glFrustum(*left, right, bottom, top, zNear, zFar*)



- Orthographic parallel projection

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



- Positioning the camera

  gluLookAt(eye_x, eye_y, eye_z,
  aim_x, aim_y, aim_z,
  up_x, up_y, up_z)

  → up vector determines unique orientation

These functions can either be called either just once, or if required can be called repeatedly upon timer events and/or keystrokes.

Modelling Transformations

- Move object

  glTranslate{fd}(*x*, *y*, *z*)

- Rotate object around arbitrary axis

  glRotate{fd}(*angle*, *x*, *y*, *z*)

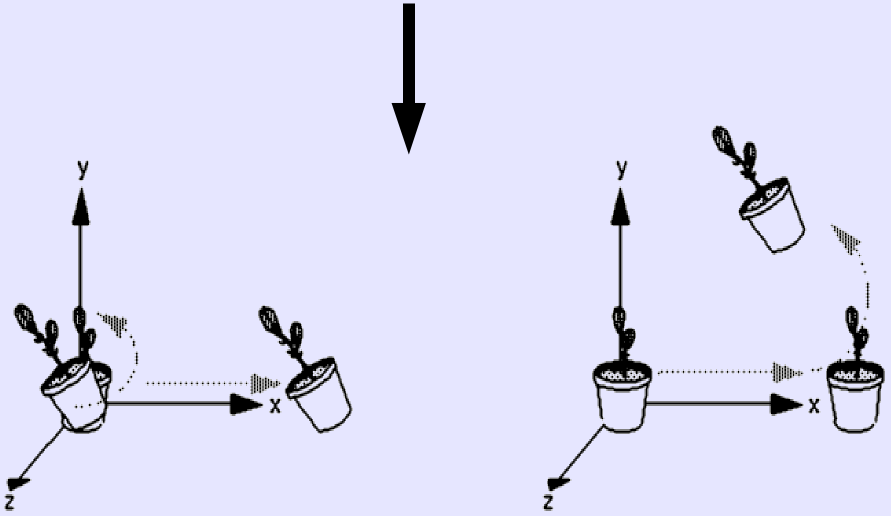
  - angle is in degrees

- Dilate (stretch or shrink) or mirror object

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

Warning:

Transformations are not commutative!



Matrix Operations

- Specify Current Matrix Stack

  glMatrixMode(GL_MODELVIEW or GL_PROJECTION)

- Other Matrix or Stack Operations

  glLoadIdentity() glPushMatrix() glPopMatrix()

- Transformations can be specified either as operations (glTranslate(), glRotate()) or as Matrices (glLoadMatrix(), glMultMatrix()).

- Viewport
  - usually same as window size
  - viewport aspect ratio should be same as projection transformation or resulting image may be distorted

  glViewport(x, y, width, height)

Manipulating the Matrix Stacks

```
draw_wheel_and_bolts()
{
 long i;

 draw_wheel();
 for(i=0;i<5;i++){
  glPushMatrix();
    glRotatef(72.0*i,0.0,0.0,1.0);
    glTranslatef(3.0,0.0,0.0);
    draw_bolt();
  glPopMatrix();
 }
}
```

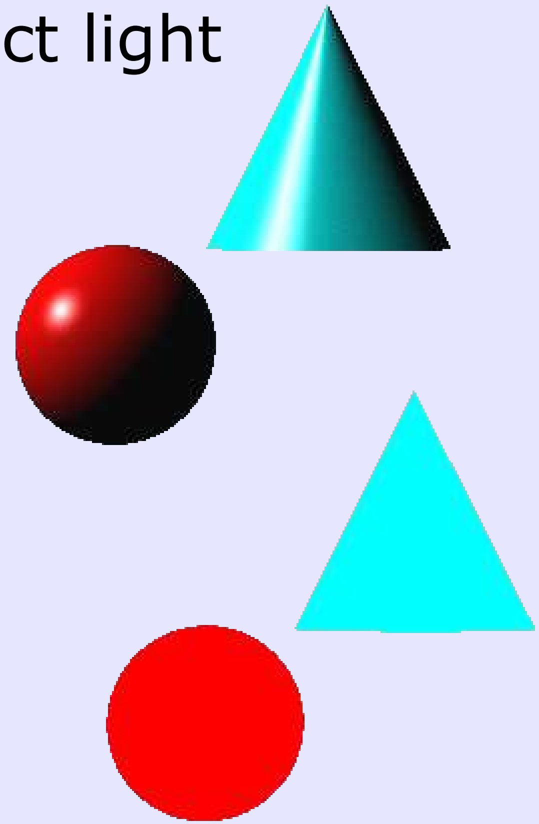
```
draw_body_and_wheel_and_bolts()
{
   draw_car_body();
   glPushMatrix();
       /*move to first wheel position*/
      glTranslatef(40,0,30);
      draw_wheel_and_bolts();
   glPopMatrix();
   glPushMatrix();
       /*move to 2nd wheel position*/
      glTranslatef(40,0,-30);
      draw_wheel_and_bolts();
   glPopMatrix();
   ...
   /*draw last two wheels similarly*/
}
```

Depth Buffer (z-Buffer)

- Hidden Surface removal is disabled by default !!!
- For each pixel on the screen, the depth buffer stores the distance between the viewpoint and the object occupying that pixel.
- A new candidate color for that pixel is drawn only if its z-value is lower!
- Enable using glEnable(GL_DEPTH_TEST) once, and glClear(GL_DEPTH_BUFFER_BIT) before every frame.

Lightining Principles

- Lighting simulates how objects reflect light
  - material composition of object
  - light’s color and position
  - global lighting parameters
    - ambient light
    - directed lighting
  - available in both color index and RGBA mode



Material and Light

- **Material Properties**

glMaterialfv(*face, property, value*);

Properties can be GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_AMBIENT_AND_DIFFUSE

Possible to specify seperately for front and back!

- **Light Properties**

glLightfv(*light, property, value*);

- light specifies which light (GL_LIGHT0 .. GL_LIGHT7)
- Properties are color, position, type and attenuation
- Light type can be Ambient, Diffused, or Specular !

Example: glLightfv(GL_LIGHT0, GL_POSITION, light_position)

Lighting must be enable using glEnable(GL_LIGHTING).

Texture Mapping

- With Texture Mapping, one applies Image Primitives (Bitmaps, Images) onto Geometric primitives (Polygons)

- OpenGL has to be told of only specific points, all other points are calculated by interpolation!

Texture Mapping (cont.)

- Textures are usually rectangular arrays of data
- Individual values called Texels
- Steps in Texture Mapping:
  1. Create a Texture Object
  2. Indicate how the texture is to be applied
  3. Enable Texture Mapping, `glEnable(GL_TEXTURE_2D)`
  4. Draw the scene specifying both texture and geometric coordinates

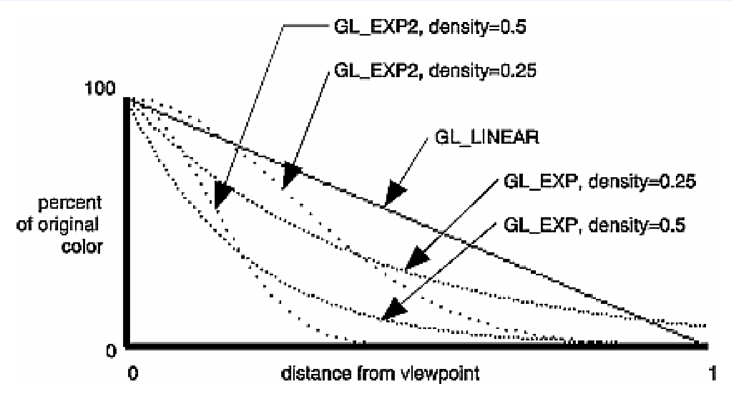
Texture Mapping (cont.)

```
 // Loading a Texture, specifying properties

 glGenTextures(1, &texName);

glBindTexture(GL_TEXTURE_2D, texName);

 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
                 width, height, 0,
                 GL_RGBA, GL_UNSIGNED_BYTE, Image_Buffer);

...... // later while describing polygons...
 glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
```

Fog

- makes objects fade into a distance
- general term for many atmospheric effects (haze, mist, smoke or pollution.
- enable fog simply by glEnable(GL_FOG)
- Fog blending factor changes by distance of object: either GL_EXP, GL_EXP2, GL_LINEAR

```
{
   GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};

   fogMode = GL_EXP;
   glFogi (GL_FOG_MODE, fogMode);
   glFogfv (GL_FOG_COLOR, fogColor);
   glFogf (GL_FOG_DENSITY, 0.35);
   glHint (GL_FOG_HINT, GL_DONT_CARE);
   glFogf (GL_FOG_START, 1.0);
   glFogf (GL_FOG_END, 5.0);
}
glClearColor(0.5, 0.5, 0.5, 1.0);  /* fog color */
```

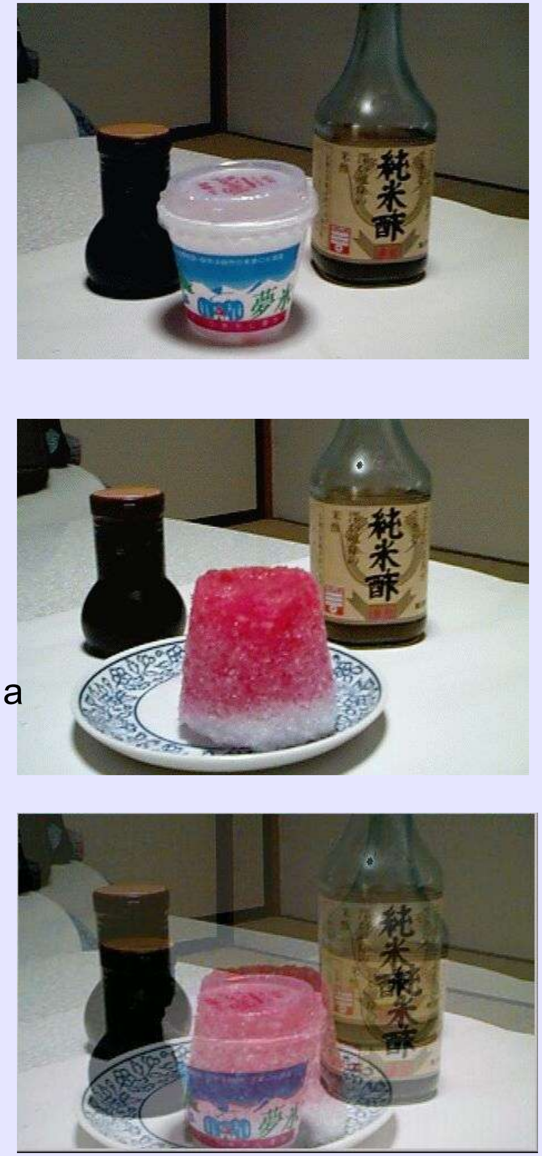


Alpha Blending

- For Creating Translucent objects
  e.g. Pilots front pane, special effects...

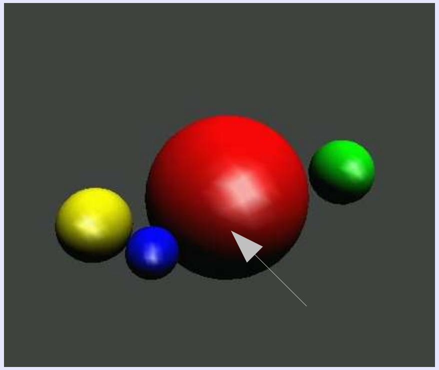
-
  ```
  glEnable(GL_BLEND);
  glBlendFunc(  GL_SRC_ALPHA,
                GL_DST_ALPHA);
  ```

- Computed color is then
  $R_sS_r+R_dD_r, G_sS_g+G_dD_g, B_sS_b+B_dD_b, A_sS_a+A_dD_a$



User Interaction

Typical Problem: How to find the object lying beneath the mouse cursor ?



- Objects drawn on the screen have typically undergone multiple rotations, translations and perspective transformations

Solution: Use Selection and Picking modes provided by OpenGL

- Returns small list of objects that are drawn near the cursor
- Objects must be first named using glPushName() and glLoadName()

Adding Complexity to Program

New Issues with large programs:

- Large number of objects
- Pre-generated models
- User Input, event handling

➔ Third-party file format loaders available for OpenGL

Display Lists

- Cache for OpenGL commands for later execution
- May improve performance

```
 // Create a new List (just once)
{
   theTorus = glGenLists (1);
   glNewList(theTorus, GL_COMPILE);
   torus(8, 25);
   glEndList();
}

// Call the List
   glCallList (theTorus);
```

OpenPerformer by SGI

- Object-oriented high-level API on top of OpenGL
- Supports model loading, OO function binding, inbuilt mouse and keyboard event handling
- Free download
- Available for IRIX, Windows and Linux

OpenGL under Linux

1) Mesa3D
  - Software implementation, non-accelerated
  - Open Source (XFree86-compatible licence)
  - Not officially OpenGL compatible

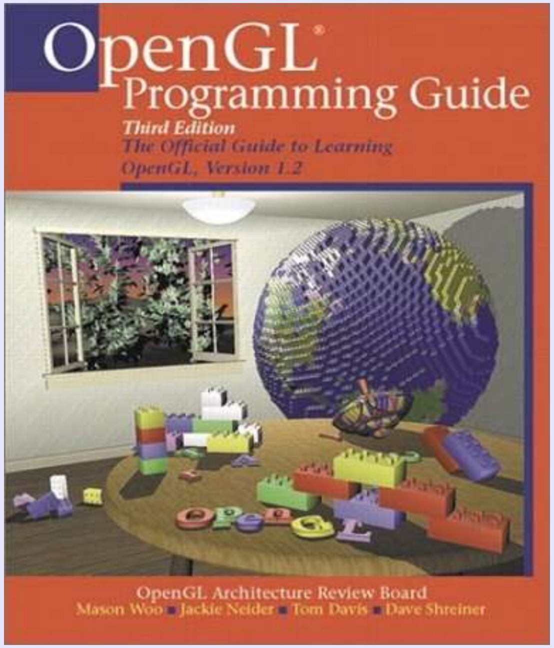
2) NVidia OpenGL driver
  - available also for Linux
  - Hardware-acceleration (on Nvidia's graphic cards like GeForce)
  - High-end performance: challenges even expensive SGI-graphic workstations

Literature

OpenGL Programming Guide, 3rd Edition
Woo, Neider, Davis, Shreiner
Addison-Wesley

3D Computer Graphics
Alan Watt
Addison-Wesley

Introduction to Computer Graphics
Foley, van Dam, ...
Addison-Wesley



- http://www.opengl.org/developers/documentation/
- http://www.3dsource.de/faq/index.htm
  OpenGL FAQ (deutsch)
- http://members.net-tech.com.au/alaneb/opengl.html
  Al's OpenGL Page