CDS 301

Fernando Camelli

Fall 2008

Some Information

- Fernando Camelli, Assistant Professor

- Web:

  www.cds.gmu.edu/~fcamelli

Recommended Books

- Visualization Handbook
  Edited by Charles D. Hansen, Chris R. Johnson
- Visualization and Processing of Tensor Fields
  Edited by Joachim Weickert, Hans Hagen
- Scientific Visualization: The Visual Extraction of Knowledge from Data
  By Georges-Pierre Bonneau, Thomas Ertl, Gregory M. Nielson
- Computer Graphics: Principles and Practice (2nd edition)
  James D. Foley, Andries Van Dam, Steven K. Feiner, John F. Hughes
- Geometric Modeling for Scientific Visualization
  Guido Brunnett, Bernd Hamann, Heinrich Müller, Lars Linsen
- Geometric Data Structures for Computer Graphics
  By Elmar Langetepe, Gabriel Zachmann
- OpenGL® Programming Guide: The Official Guide to Learning OpenGL®, Version 2.1
  By OpenGL Architecture Review Board, Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis
- OpenGL® SuperBible: Comprehensive Tutorial and Reference
  By Richard S. Wright, Benjamin Lipchak, Nicholas Haemel
- Python Scripting for Computational Science
  By Hans Petter Langtangen

Useful Links

- OpenGL
  - http://www.opengl.org/
  - http://www.starstonesoftware.com/OpenGL/
- GLUT
  - http://freeglut.sourceforge.net/

- GUI
  - OpenMotif
    - http://www.opengroup.org/openmotif/
  - GTK
    - http://www.gtk.org/
  - Qt
    - http://trolltech.com/products/qt
    - http://www.qtcentre.org/
    - http://qtnode.net/wiki?title=Main_Page

More Useful Links

- Python
  - http://www.python.org/
  - http://folk.uio.no/hpl/scripting/
- VTK and ParaView
  - http://www.vtk.org/
- ITK
  - http://www.itk.org/

Outline

- Part 1: Computer Graphics
  - Windows Microsoft and Linux
  - Introduction to OpenGL
  - OpenGL drawing techniques
  - GLUT, OpenMotif, Qt, GTK – How to open a window with a drawing area
  - Event management

- Part 2: Advanced Computer Graphics
  - Graphics pipeline
  - Mouse control
  - Display list
  - Shading, illumination, texture

Part 1

Computer Graphics

Environment

- Linux: software
  - OpenGL
  - GLUT
  - OpenMotif, Qt, GTK
- Windows
  - Cygwin
  - Visual Studio
  - Qt, GTK
- All my examples are tested in Linux or Microsoft Windows using Cygwin

Cygwin

- Web
  - http://cygwin.com/
- Download setup.exe
  - http://cygwin.com/setup.exe
- Install
  - All Devel components
  - All X11components
  - All Math components
  - All Lib components
  - And if you would like to be sure, install everything

Some Questions

- C
- C++
- Another programming language?
- Linux
- Windows
- Another operating system?

- OpenGL
- GLUT
- OpenMotif
- Qt
- GTK

Computer Graphics

- Three dimensional representation (3D)
- Objects
  - width + height + depth
- Two dimensional (2D) + perspective
  - Creates appearance of three dimensions

3D Effects

- Perspective
  - Angles between lines – illusion of 3D
- Color and Shading
- Light and shadows
  - Give different colors depending the position of the light source
- Texture mapping
  - Realism and detail
- Fog
  - Haziness, it is related to distance to the observer
- Blending and transparency
  - Combining colors
- Antialiasing
  - Discrete pixels, jagged edges

Example 3D Effects

- Tutorial from “OpenGL SuperBible”
  - http://www.starstonesoftware.com/OpenGL/

- Cube (open example)
  - Perspective
  - Color and Shading
  - Light and shadows
  - Texture mapping

OpenGL

- 3D graphics library
- Initially developed by SGI
  - First: IRIS GL
- It is not a language
  - Application Programming Interface (API)
- 3D API standard
  - OpenGL Architecture Review Board (ARB)
- OpenGL is supported by most of the Operating Systems (OS)

OpenGL does not …

- Open a window (where we draw objects)
  - We need
    - GLUT, Motif, Qt, GTK,…
  - Windows and Unix/Linux/Mac
    - Different models to handle the drawing area
- Manage event driven model
  - Mouse control
  - Resizing, focus, …

Data Types

• GLbyte	8-bit integer (b)
• GLshort	16-bit integer (s)
• GLint,GLsizei	32-bit integer (l)
• GLfloat,GLclampf	32-bit floating point (f)
• GLdouble,GLclampd	64-bit floating point (d)
• GLubyte,GLboolean	8-bit unsigned integer (ub)
• GLushort	16-bit unsigned integer (us)
• GLuint,GLenum,GLbitfield	32-bit unsigned integer (ui)
• GLchar	8-bit character (none)
• GLsizeptr,GLintptr	native pointer (none)

OpenGL Headers

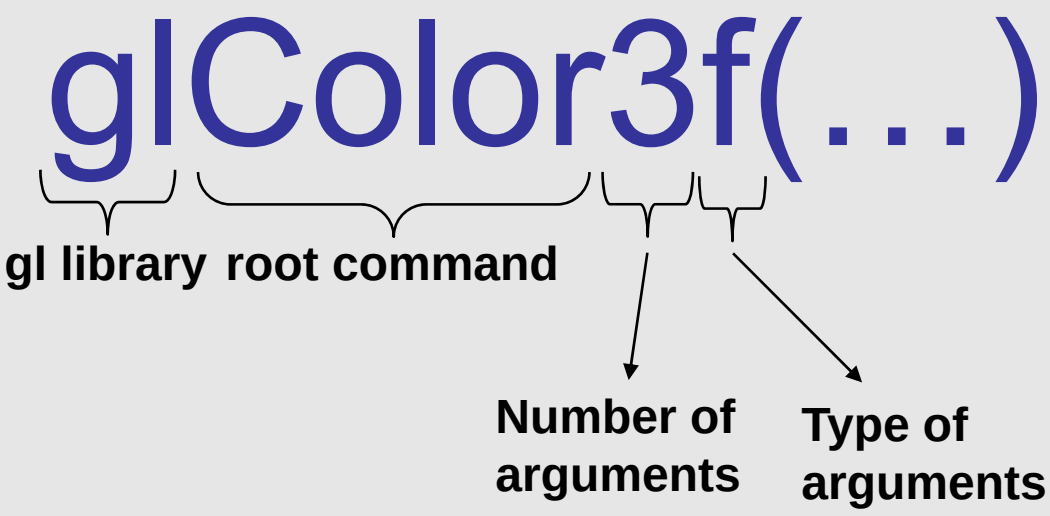
- Windows
  - #include <windows.h> /* needs to be before gl.h and glu.h */
  - #include <gl\gl.h> /* OpenGL Headers for Microsoft */
  - #include <gl\glu.h> /* OpenGL Utilities */

- Linux – Unix
  - #include <GL/gl.h>
  - #include <GL/glu.h>

```
#if defined(_LINUX_)
#include <GL/gl.h>
#include <GL/glu.h>
#elif defined(_WINDOWS_)
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#endif
```

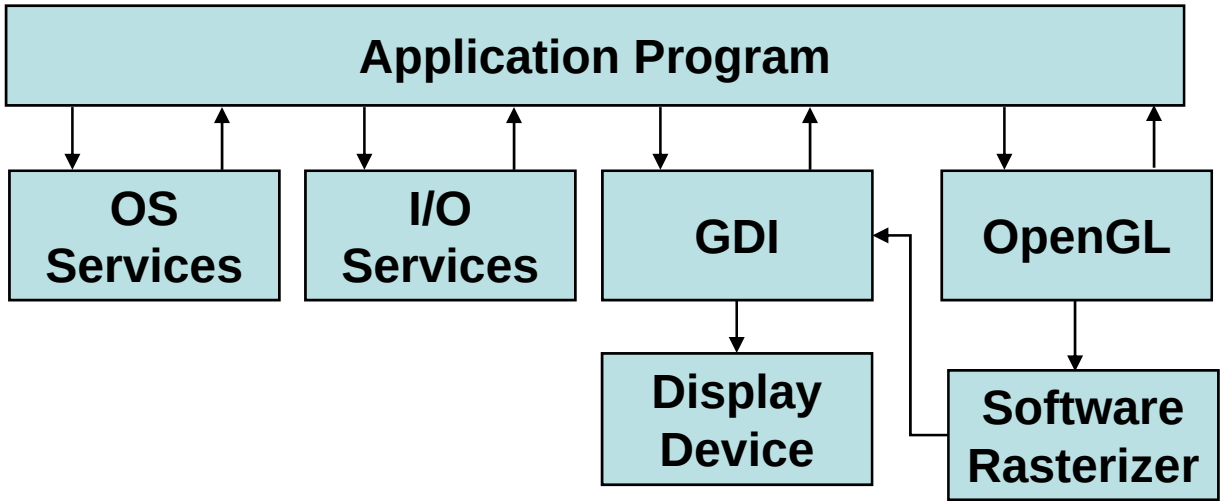
Function-Naming Convention

<library prefix><root command><optional argument count><optional argument type>

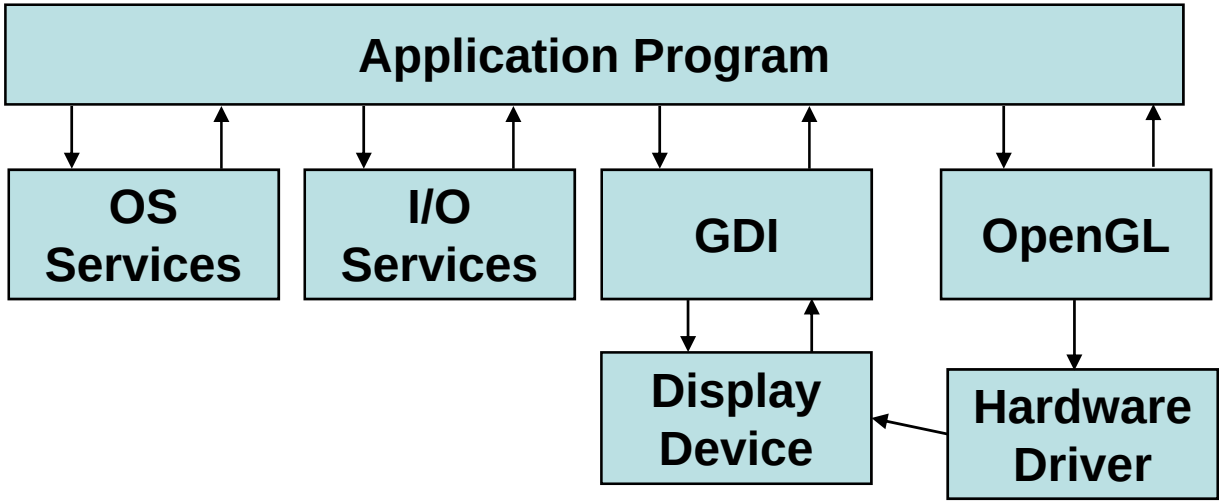


Hardware and Software

Generic



Hardware



Pipeline

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix} \Rightarrow \begin{bmatrix} \textbf{Modelview} \\ \textbf{matrix} \end{bmatrix} \Rightarrow \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} \Rightarrow \begin{bmatrix} \textbf{Projection} \\ \textbf{matrix} \end{bmatrix} \Rightarrow \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \Rightarrow \cdots$$

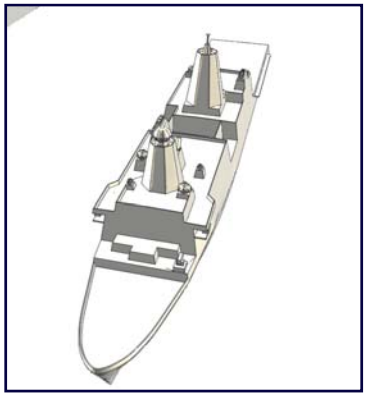
Original vertex data

Transformed eye coordinates

Clip coordinates

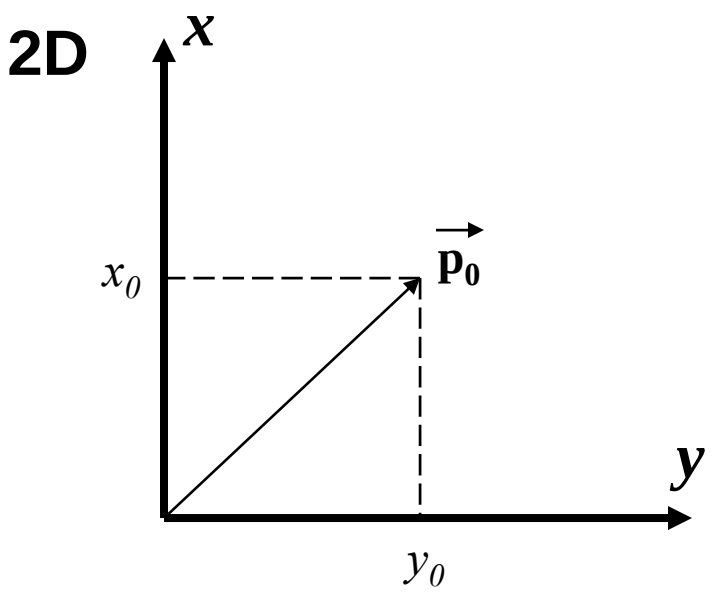
$$\cdots \Rightarrow \boxed{\textbf{Perspective division}} \Rightarrow \begin{bmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \textbf{Viewport} \\ \textbf{transformation} \end{bmatrix} \Rightarrow$$

Normalized device coordinates

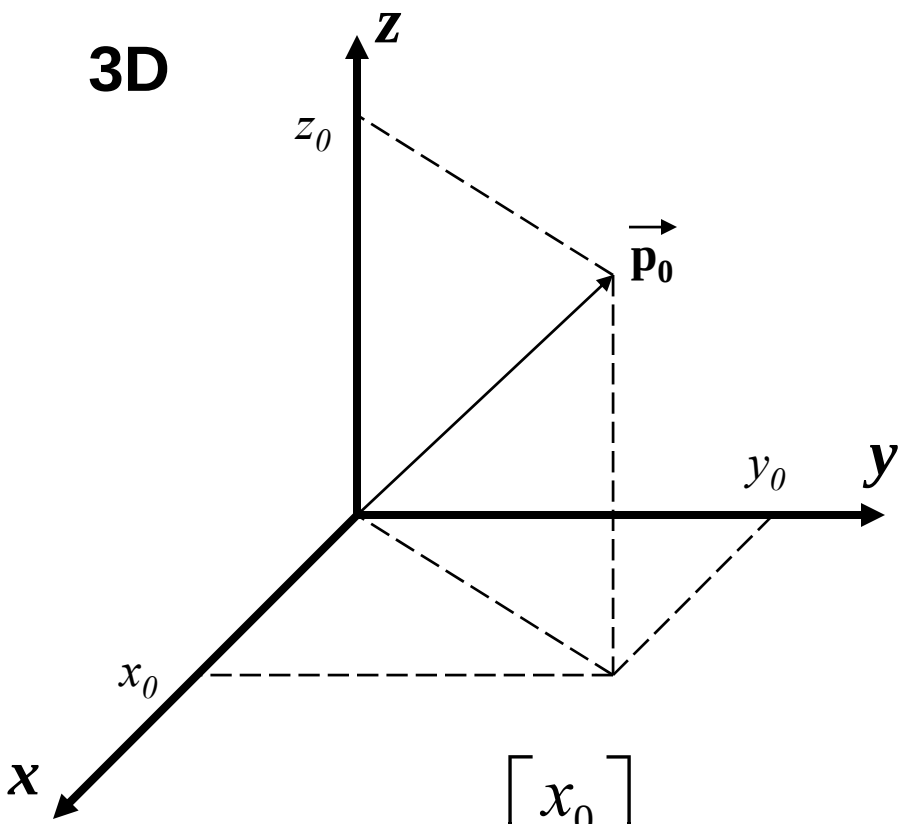


Window coordinates

Coordinates in 2D and 3D



$$\vec{\mathbf{p}}_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

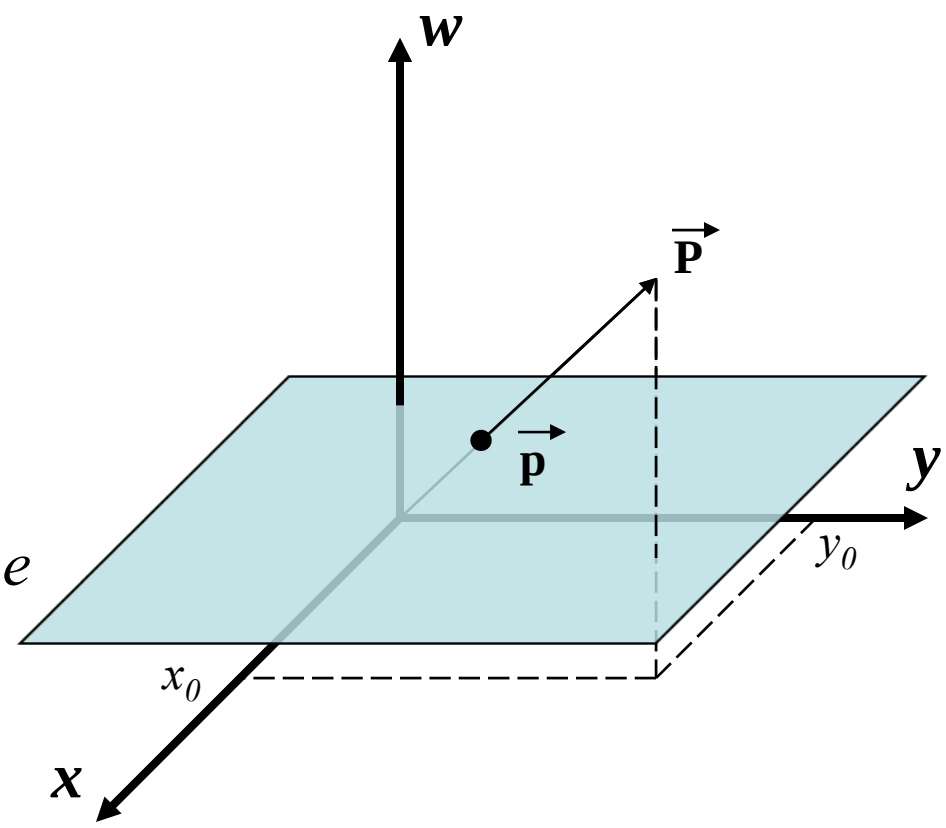


$$\vec{\mathbf{p}}_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

Homogeneous Coordinates in 2D

$$\begin{bmatrix} X_0 \\ Y_0 \\ w \end{bmatrix}, \textbf{for any } w \neq 0, \begin{bmatrix} X_0 / w \\ Y_0 / w \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

$$\vec{\mathbf{p}} = \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

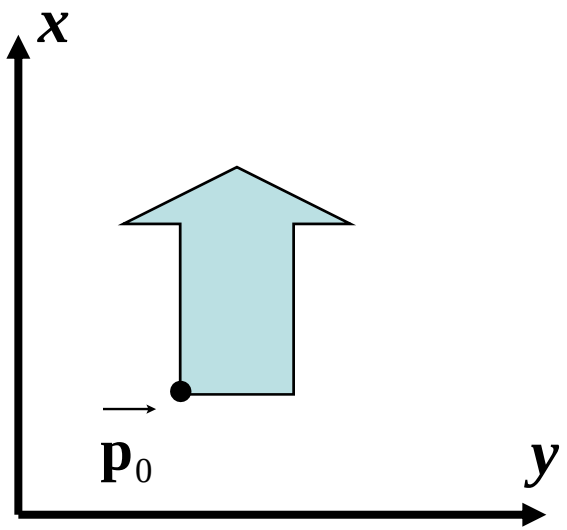


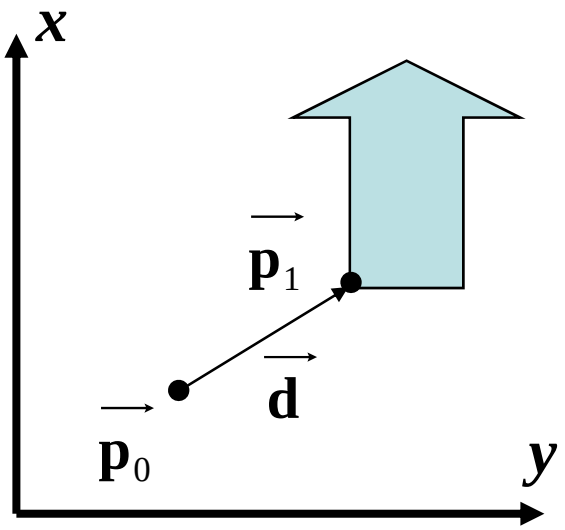
w = 1 plane

Modeling Transformations

- Translate
- Scale
- Rotate

Translate

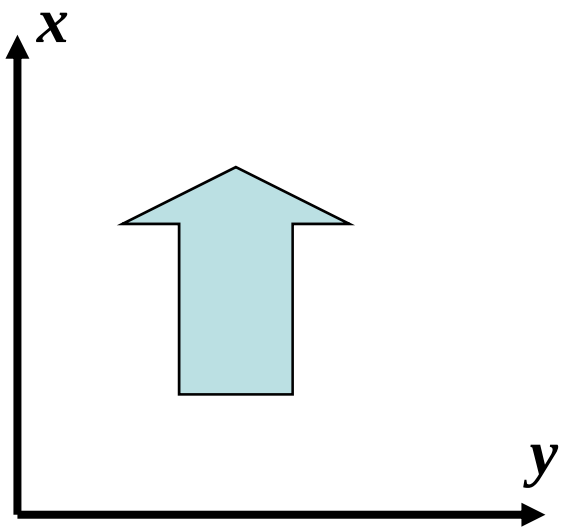




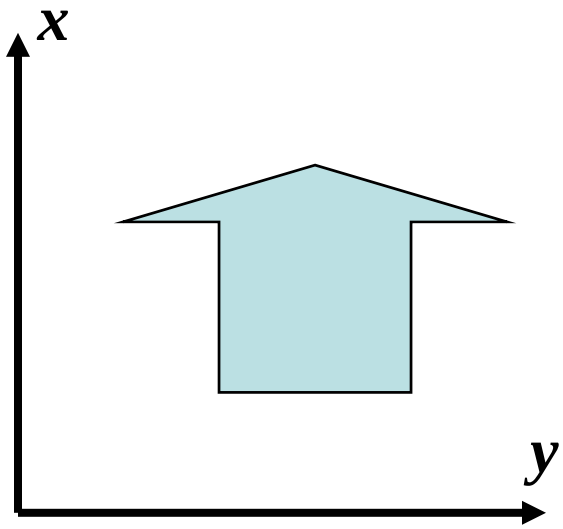
$$\overrightarrow{\mathbf{p_0}} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

$$\overrightarrow{\mathbf{p}_1} = \overrightarrow{\mathbf{p}_0} + \vec{\mathbf{d}} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

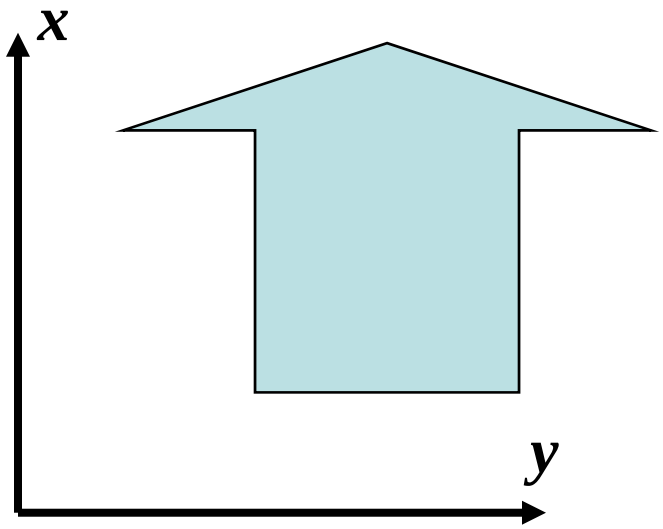
Scale



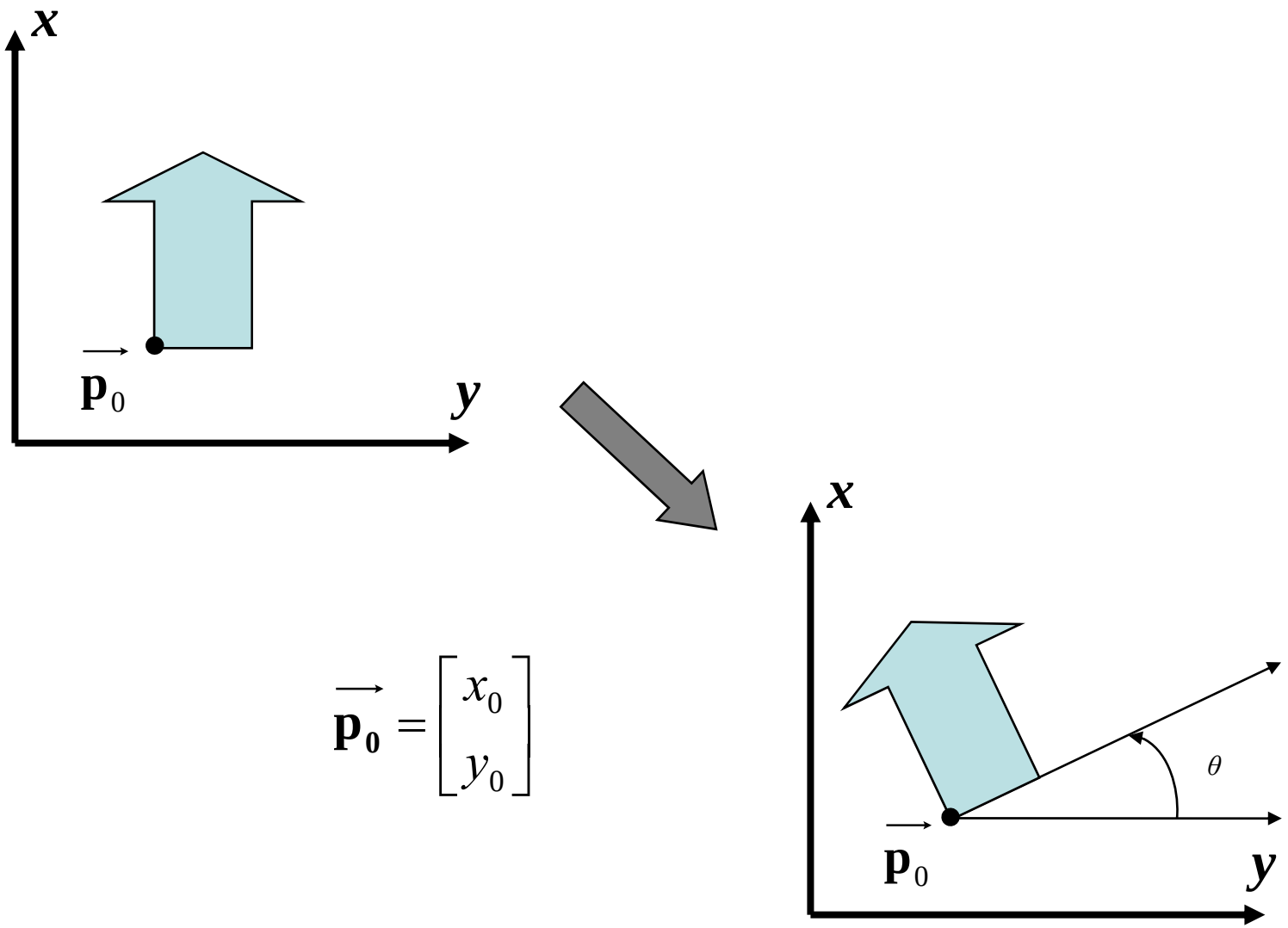
Scaling in y



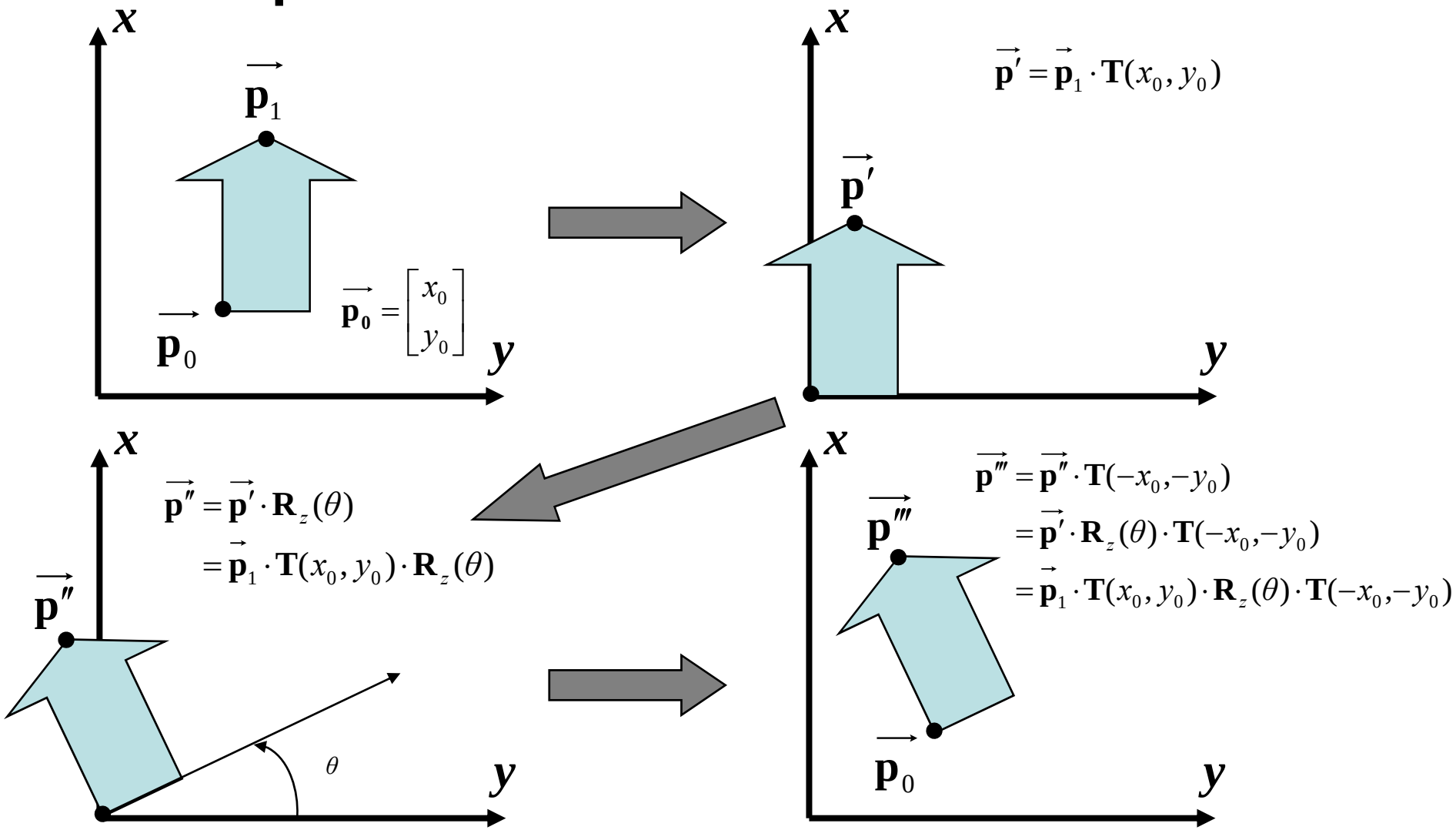
Scaling in x and y



Rotate


x
y
$\mathbf{p}_0$
$\overrightarrow{\mathbf{p}_0} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$
x
y
$\mathbf{p}_0$
θ

Composition of Transformations



Translate in 3D – Matrix

$$\mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \vec{\mathbf{p}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\vec{\mathbf{p}'} = \mathbf{T}(d_x, d_y, d_z) \cdot \vec{\mathbf{p}}$$

$$\vec{\mathbf{p}'} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{bmatrix}$$

Scale in 3D – Matrix

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \vec{\mathbf{p}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\vec{\mathbf{p}'} = \mathbf{S}(s_x, s_y, s_z) \cdot \vec{\mathbf{p}}$$

$$\vec{\mathbf{p}'} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ 1 \end{bmatrix}$$

Rotate in 3D – Matrices

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\vec{\mathbf{p}}' = \mathbf{R}_{axis}(\theta) \cdot \vec{\mathbf{p}}$$

$$\vec{\mathbf{p}}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y\cos\theta - z\sin\theta \\ y\sin\theta + z\cos\theta \\ 1 \end{bmatrix}$$

Modeling and Viewing Transformations

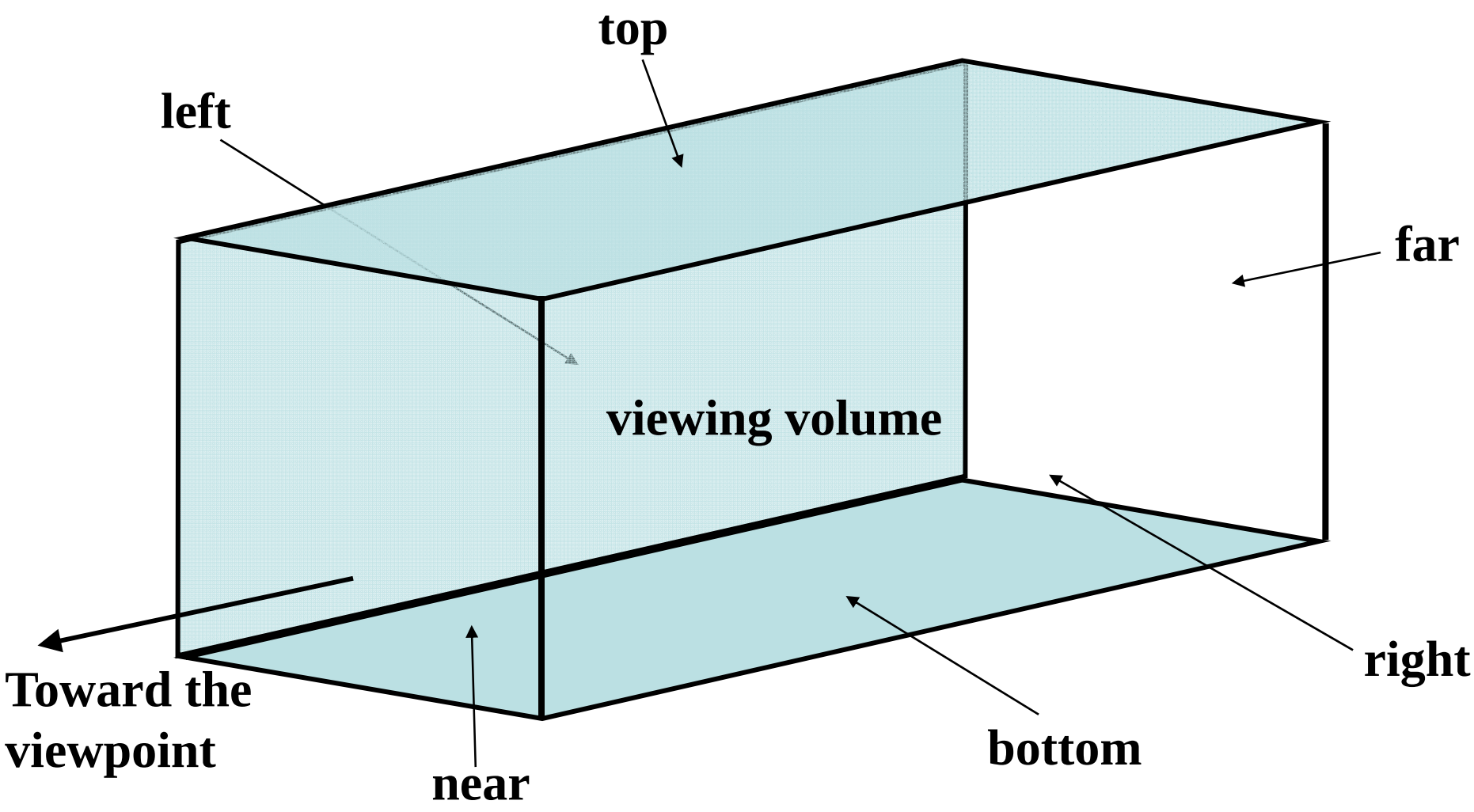
- Translation
  - void glTranslate*(TYPE *x*, TYPE *y*, TYPE *z*);
- Rotate
  - void glRotate*(TYPE *angle*, TYPE *x*, TYPE *y* , TYPE *z*);
- Scale
  - void glScale*(TYPE *x*, TYPE *y* , TYPE *z*);

- Line of sight
  - void gluLookAt(GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*, GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*, GLdouble *upx*, GLdouble *upy*, GLdouble *upz*);

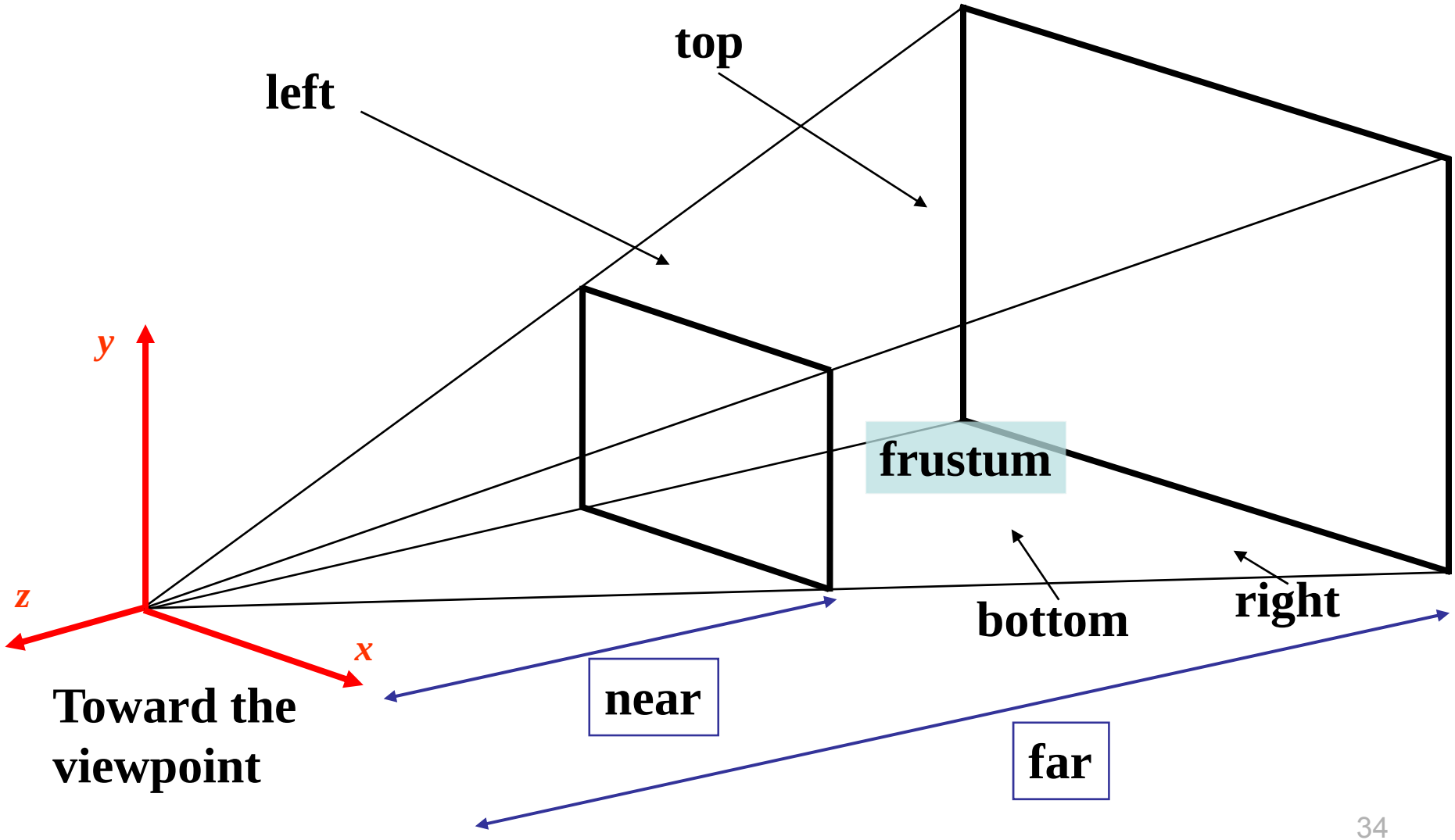
Projection Transformations

- Perspective projection
- Orthographic projection
- Example of perspective and orthographic projection
  - Open examples:
    - cube_persp.c
    - cube_ortho.c

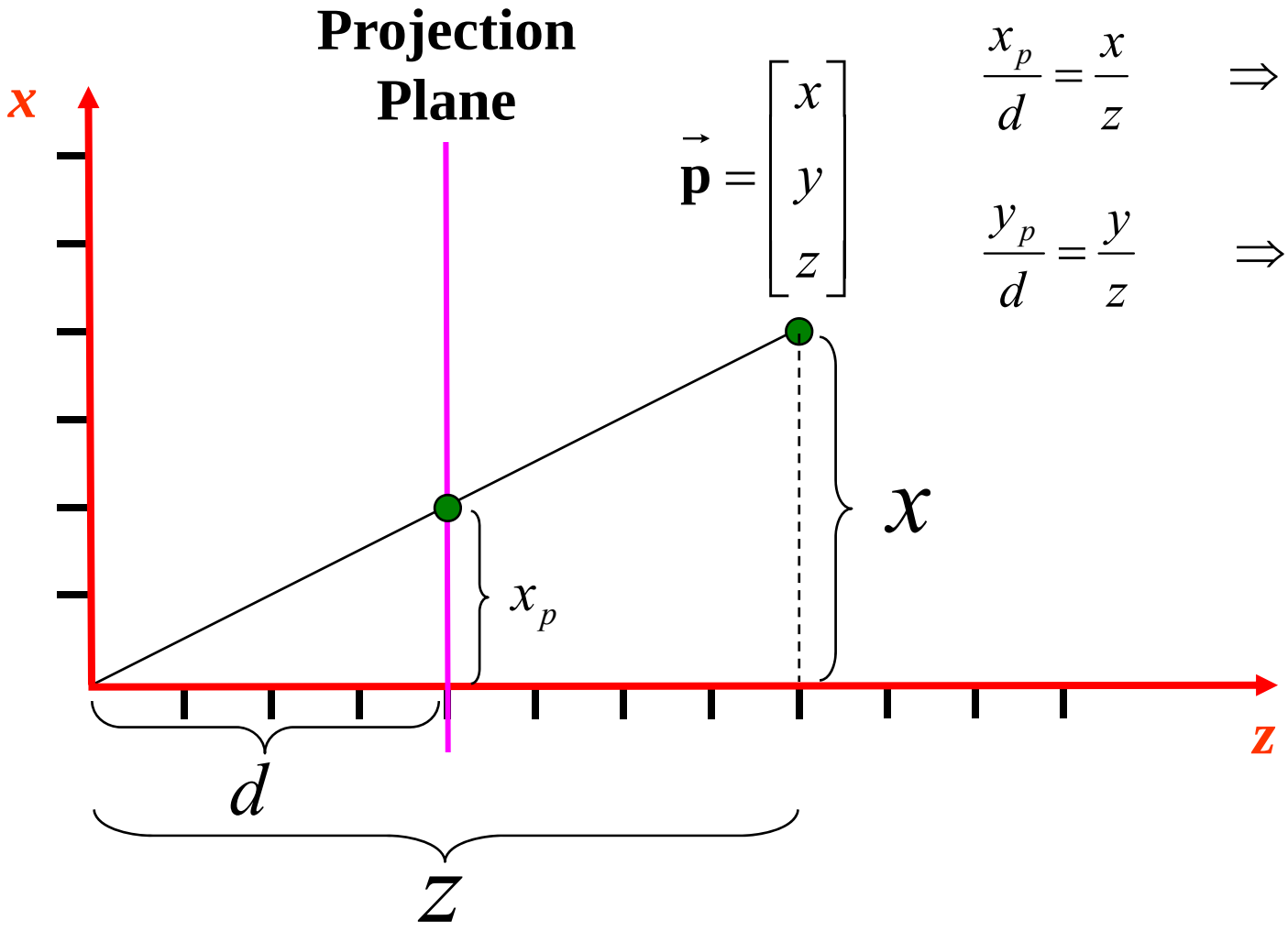
Orthographic Projection



Perspective Projection



Perspective Projection Idea



$$\frac{x_p}{d} = \frac{x}{z} \quad \Rightarrow \quad x_p = \frac{x}{z/d}$$

$$\frac{y_p}{d} = \frac{y}{z} \quad \Rightarrow \quad y_p = \frac{y}{z/d}$$

Matrix Projections

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

$$\mathbf{R}^{-1}_{perspective} = \begin{bmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{-(f-n)}{2fn} & \frac{f+n}{2fn} \end{bmatrix}$$

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

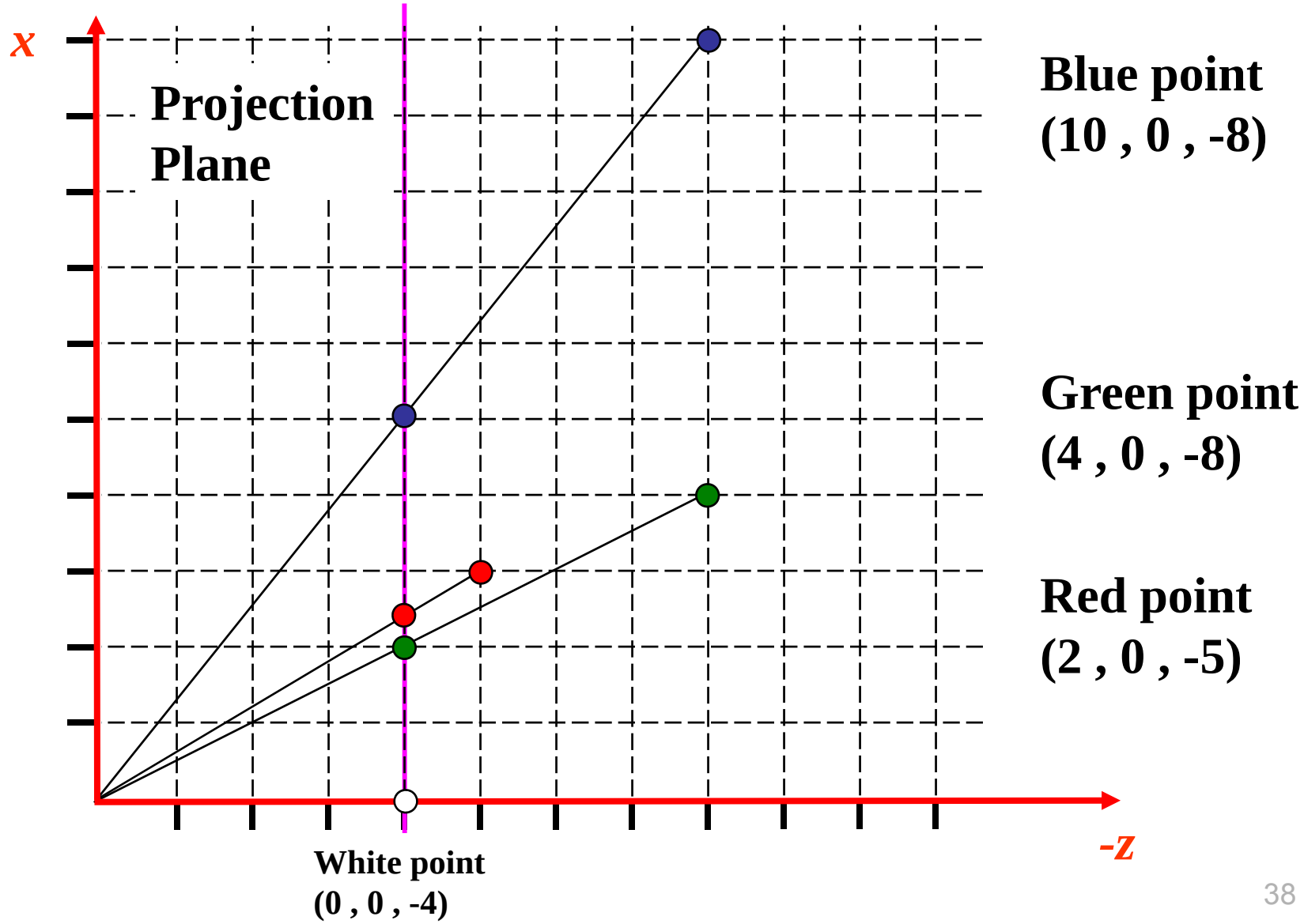
$$\mathbf{R}^{-1}_{orthographic} = \begin{bmatrix} \frac{r-l}{2} & 0 & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & 0 & \frac{t+b}{2} \\ 0 & 0 & \frac{f-n}{-2} & \frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

l : **left** ; r : **right** ; b : **bottom** ; t : **top** ; n : **near** ; f : **far**

Projection Transformations

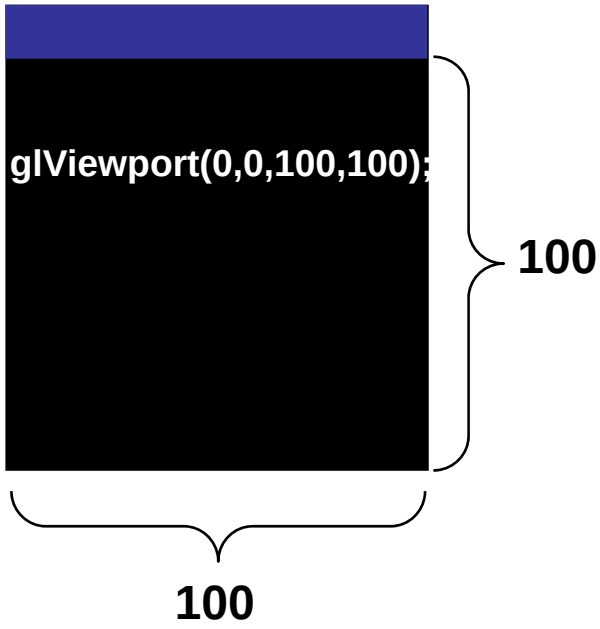
- Perspective
  - void glFrustum(GLdouble *left*, GLdouble *right*,
    GLdouble *bottom*, GLdouble *top*,
    GLdouble *near*, GLdouble *far*);
  - void gluPerspective(GLdouble *fovy*, GLdouble *aspect*,
    GLdouble *near*, GLdouble *far*);
- Orthographic
  - void glOrtho(GLdouble *left*, GLdouble *right*,
    GLdouble *bottom*, GLdouble *top*,
    GLdouble *near*, GLdouble *far*);

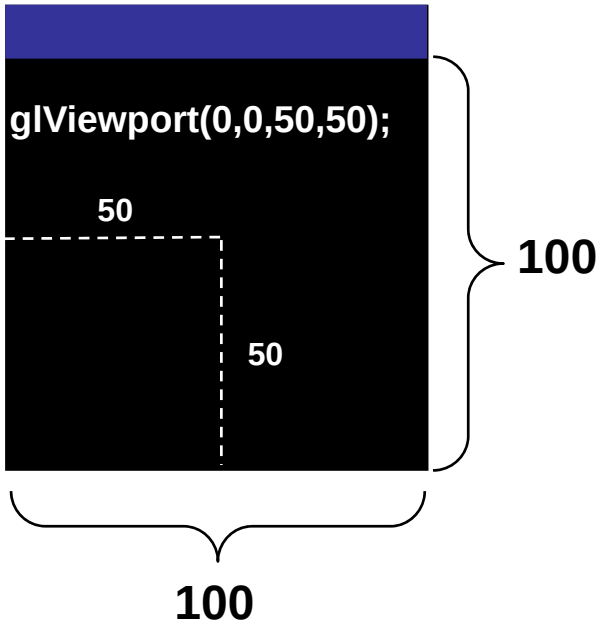
Example of Perspective Projection



Viewport

- void glViewport(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);
  - *x* and *y*: lower left corner of the window
  - *width* and *height*: dimensions in pixels
  - Viewport defines area within the window in screen coordinates
  - The clipping volume is mapped to the viewport





More Examples for Projection

- Open example using perspective matrix projection

- Open example of the three points

OpenGL Related Libraries

- OpenGL Utility Library (GLU)
- Libraries to extend the functionality of a windows system to support OpenGL
  - OpenGL Extension to the X System (GLX)
- OpenGL Utility Toolkit (GLUT)

GLUT Headers

- Windows
  - #include <gl\glut.h>
- Linux – Unix
  - #include <GL/glut.h>

```
#if defined(_LINUX_)

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

#elif defined(_WINDOWS_)

#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#endif
```

Windows Management (GLUT)

- glutInit(int **argc*,char ***argv*)
  - Initialization and process command line arguments
- glutInitDisplayMode(unsigned int *mode*)
  - Specifies wether you use RGB or a color index color model, single- or double-buffer, associate depth, stencil and/or accumulation buffer
    glutInitDisplayMode(*GLUT_DOUBLE*|*GLUT_RGB*|*GLUT_DEPTH*)
- glutInitWindowPosition(int *x*, int *y*)
  - Positions the window in screen (upper-left corner)
- glutWindowSize(int *width*, int *size*)
  - Specifies size in pixels
- glutCreateWindow(char **string*)
  - Creates the windowwith an OpenGL context

Display Callback – Running (GLUT)

- glutDisplayFunc(void (**func*)(void))
  - Register func to be executed every time we need to redisplay the window
  - Routines that redraw the scene go in func
- glutMainLoop()
  - Event processing begins with the call to this function
  - All the windows are now shown

Handling Input Events (GLUT)

- glutReshapeFunc(void (**func*)(int *w*,int *h*))
  - Action when the windows is resized

- glutKeyBoardFunc(void (**func*)
  (unsigned char *key*, int *x*, int *y*))
  glutMouseFunc(void(**func*)
  (int *button*, int *state*, int *x*, int *y*))
  - Link keyboard and mouse to an event

- glutMotionFunc(void(**func*) (int *x*, int *y*))
  - Routine that is call back when mouse is moved while a mouse button is also pressed

Drawing 3D Objects (GLUT)

- void glutWireCube(GLdouble *size*)
- void glutSolidCube(GLdouble *size*)
- void glutWireSphere(GLdouble *radius*, Glint *slices*, Glint *stacks*)
- void glutSolidSphere(GLdouble *radius*, Glint *slices*, Glint *stacks*)

Clearing Buffers

- Clearing the window to black (RGB mode)
  - glClearColor(0.0, 0.0, 0.0, 0.0); /* set clear color */
  - glClear(GL_COLOR_BUFFER_BIT); /* clear window to black */

- void glClearColor(GLclampf *red*,
  GLclampf *green*,
  GLclampf *blue*,
  GLclampf *alpha*);

- void glClear(GLbitfield *mask*);

Color buffer	GL_COLOR_BUFFER_BIT	glClearColor()
Depth buffer	GL_DEPTH_BUFFER_BIT	glClearDepth()
Accumulation buffer	GL_ACCUM_BUFFER_BIT	glClearAccum()
Stencil buffer	GL_STENCIL_BUFFER_BIT	glClearStencil()

Some OpenGL Commands

- void glColor*();
  - Specifies color in RGB mode
- void glFlush(void);
  - Forces all previously issued commands to begin execution
- void glFinish(void);
  - Forces all previously issued commands to begin execution. This command doesn't return until all commands are executed

Selecting Stack

- void glMatrixMode(GLenum *mode*);
  - Mode:
    - GL_MODELVIEW (modeling and viewing transformations)
    - GL_PROJECTION (projection transformations)
    - GL_TEXTURE
  - Specifies which matrix stack is the target for subsequent operations
- void glLoadIdentity(void);
  - Replaces the current matrix with the identity matrix

OpenGL Vertices

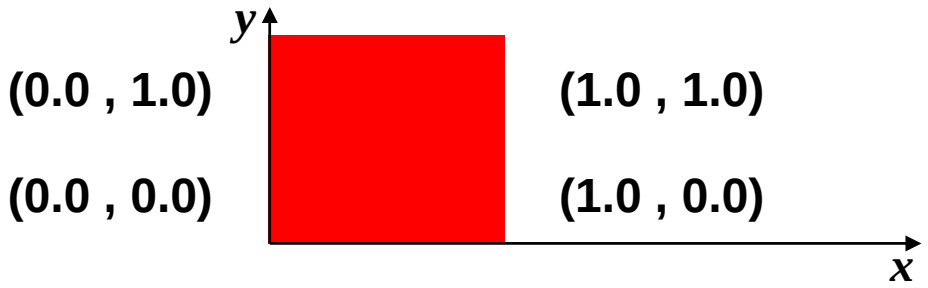
- void glVertex{2,3,4}{sifd}[v](TYPE *coords*);

```
glVertex2s( 2 , 3 );                        /* short integer: x = 2 ; y = 3 ; z = 0 */
glVertex3d( 0.0 , 0.0 , 2.3 );              /* double */
glVertex4f( 2.0 , 4.0 , 6.0 , 2.0 );        /* float: homogeneous coordinates */

/* …  */

GLdouble vect[3] = { 1.0 , 3.0 , 2.0 };     /* define a vector type double */

glVertex3dv(vect);                          /* use the vector vect */
```

Drawing Primitives

- void glBegin(GLenum *mode*);
- void glEnd(void);

```
glBegin(GL_POLYGON);                    /* GL_POLYGON: filled polygon */
        /* drawing a square */
        glVertex2f( 0.0 , 0.0 );        /* 1st vertex */
        glVertex2f( 0.0 , 1.0 );        /* 2nd vertex */
        glVertex2f( 1.0 , 1.0 );        /* 3rd vertex */
        glVertex2f( 1.0 , 0.0 );        /* 4th vertex */
glEnd();
```



Primitives – glBegin(GLenum *mode*)

- GL_POINTS — glBegin(GL_POINTS);

- GL_LINES — glBegin(GL_LINES);
- GL_LINE_STRIP — glBegin(GL_LINE_STRIP);
- GL_LINE_LOOP — glBegin(GL_LINE_LOOP);

- GL_TRIANGLES — glBegin(GL_TRIANGLES);
- GL_TRIANGLE_STRIP — glBegin(GL_TRIANGLE_STRIP);
- GL_TRIANGLE_FAN — glBegin(GL_TRIANGLE_FAN);

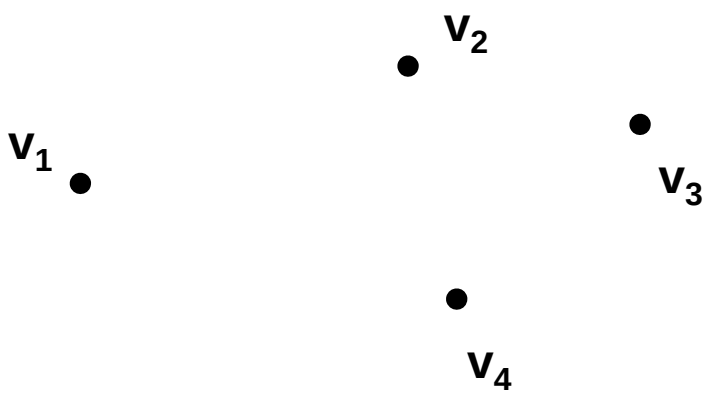
- GL_QUADS — glBegin(GL_QUADS);
- GL_QUAD_STRIP — glBegin(GL_QUAD_STRIP);

- GL_POLYGON — glBegin(GL_POLYGON);

- Do not forget glEnd()

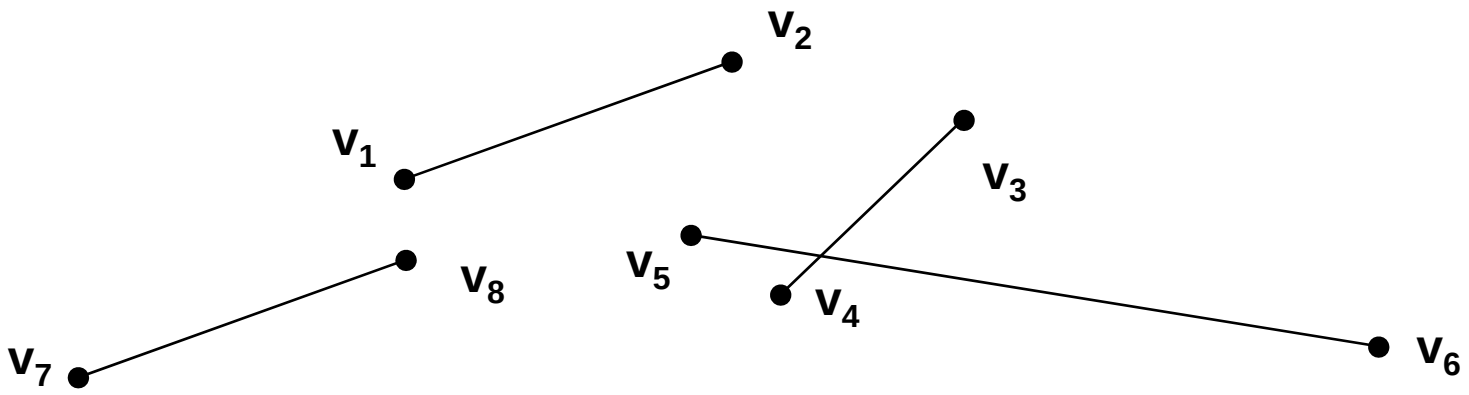
GL_POINTS

```
glBegin(GL_POINTS);
        /* drawing points */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        glVertex2f( x3 , y3 );          /* 3rd vertex */
        glVertex2f( x4 , y4 );          /* 4th vertex */
glEnd();
```



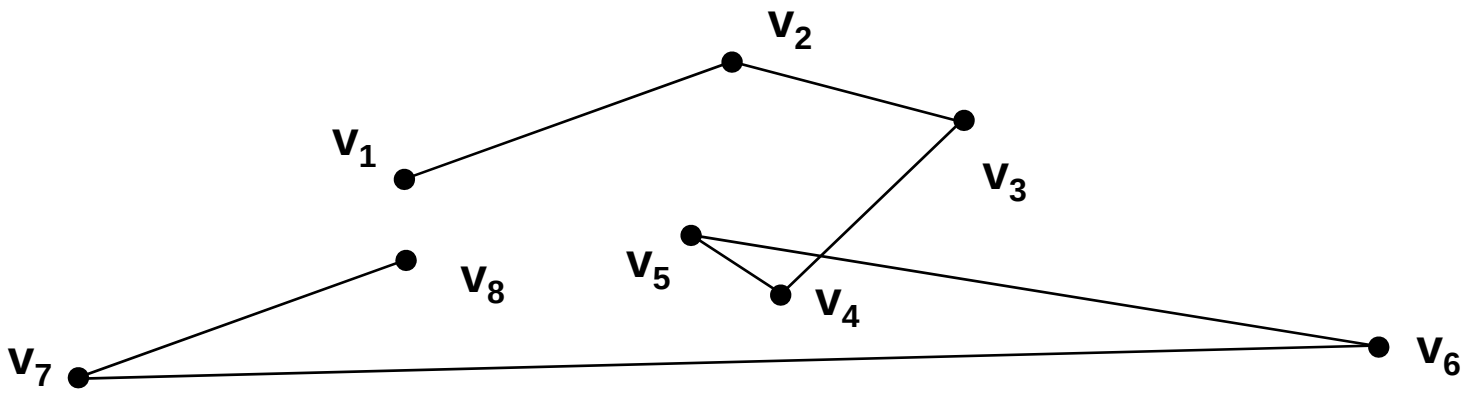
GL_LINES

```
glBegin(GL_LINES);
        /* drawing lines */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x7 , y7 );          /* 7th vertex */
        glVertex2f( x8 , y8 );          /* 8th vertex */
glEnd();
```



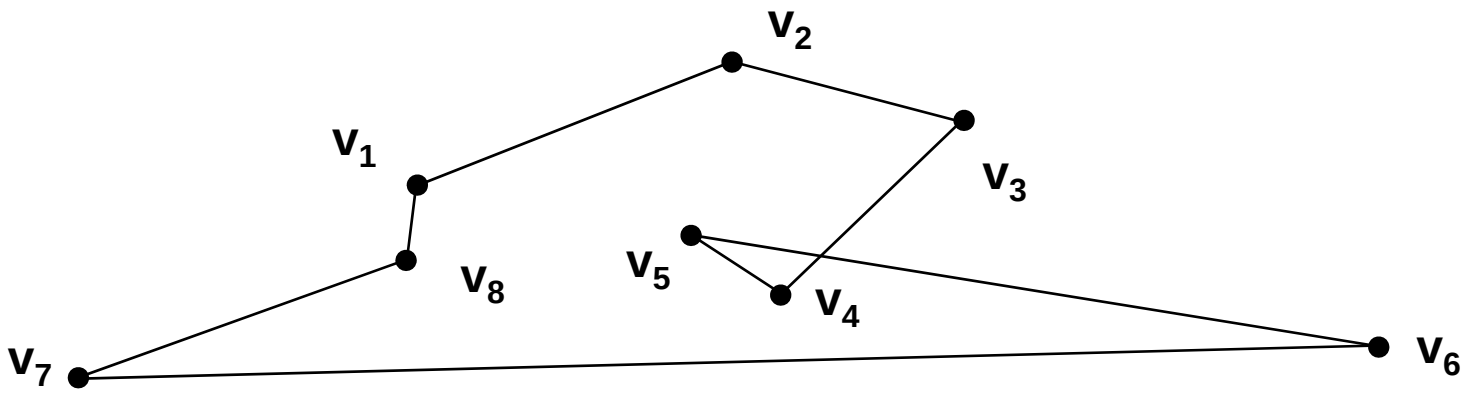
GL_LINE_STRIP

```
glBegin(GL_LINE_STRIP);
        /* drawing line strip */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x7 , y7 );          /* 7th vertex */
        glVertex2f( x8 , y8 );          /* 8th vertex */
glEnd();
```



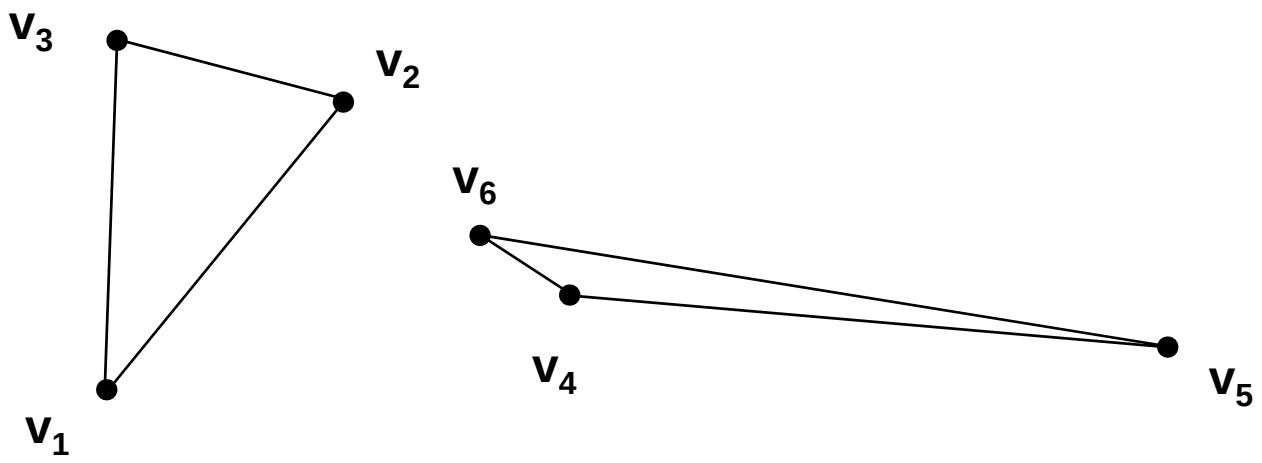
GL_LINE_LOOP

```
glBegin(GL_LINE_LOOP);
        /* drawing line loop */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x7 , y7 );          /* 7th vertex */
        glVertex2f( x8 , y8 );          /* 8th vertex */
glEnd();
```



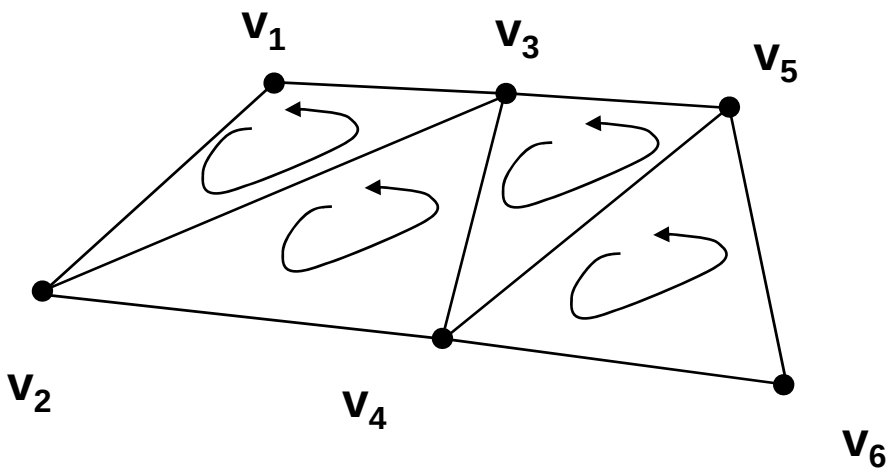
GL_TRIANGLES

```
glBegin(GL_TRIANGLES);
        /* drawing triangles */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x5 , y5 );          /* 5th vertex */
        glVertex2f( x6 , y6 );          /* 6th vertex */
glEnd();
```



GL_TRIANGLE_STRIP

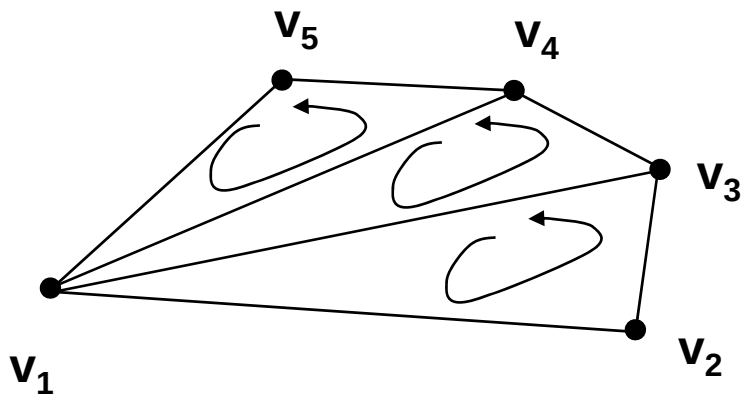
```
glBegin(GL_TRIANGLE_STRIP);
        /* drawing triangle strip */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x5 , y5 );          /* 5th vertex */
        glVertex2f( x6 , y6 );          /* 6th vertex */
glEnd();
```



Triangle 1	$v_1\ v_2\ v_3$
Triangle 2	$v_3\ v_2\ v_4$
Triangle 3	$v_4\ v_5\ v_3$
Triangle 4	$v_4\ v_6\ v_5$

GL_TRIANGLE_FAN

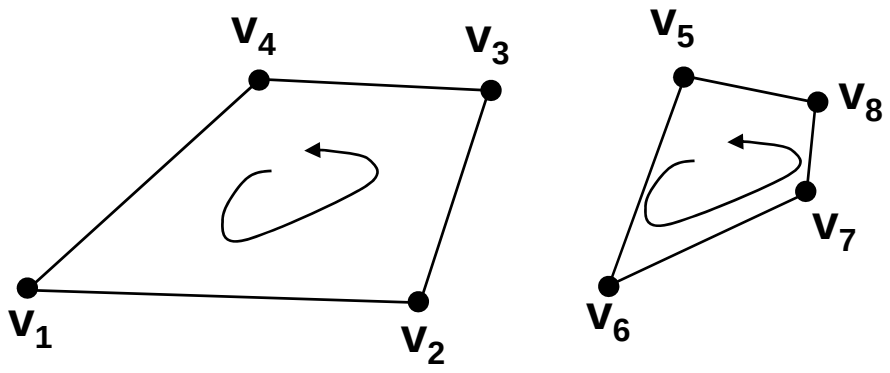
```
glBegin(GL_TRIANGLE_FAN);
        /* drawing triangle fan */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x5 , y5 );          /* 5th vertex */
glEnd();
```



Triangle 1	$v_1\ v_2\ v_3$
Triangle 2	$v_1\ v_3\ v_4$
Triangle 3	$v_1\ v_4\ v_5$

GL_QUADS

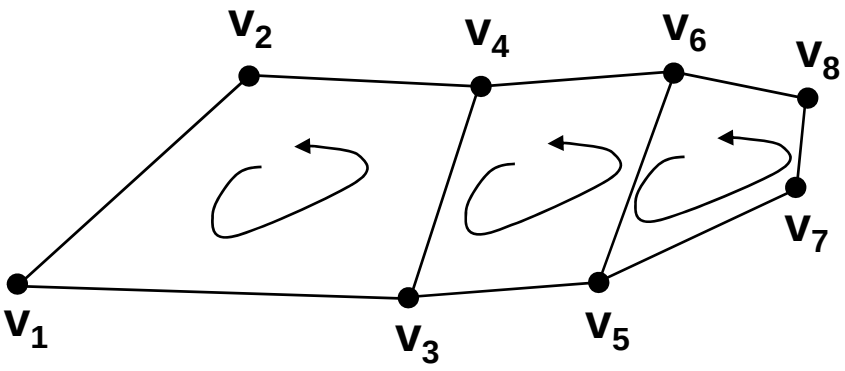
```
glBegin(GL_QUADS);
        /* drawing quads */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x8 , y8 );          /* 8th vertex */
glEnd();
```



Quad 1 $v_1\ v_2\ v_3\ v_4$
Quad 2 $v_5\ v_6\ v_7\ v_8$

GL_QUAD_STRIP

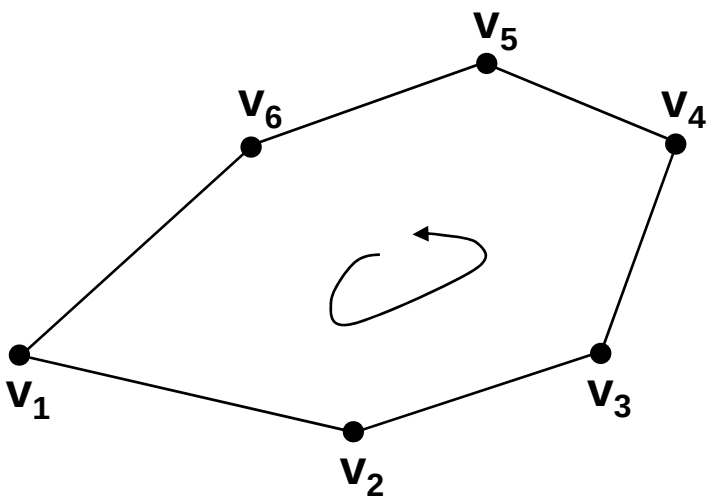
```
glBegin(GL_QUAD_STRIP);
        /* drawing quad strip */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x8 , y8 );          /* 8th vertex */
glEnd();
```



Quad 1 $v_1\ v_3\ v_4\ v_2$
Quad 2 $v_5\ v_6\ v_7\ v_8$

GL_POLYGONS

```
glBegin(GL_POLYGONS);
        /* drawing polygons */
        glVertex2f( x1 , y1 );          /* 1st vertex */
        glVertex2f( x2 , y2 );          /* 2nd vertex */
        /* … */
        glVertex2f( x6 , y6 );          /* 6th vertex */
glEnd();
```



Polygon 1 $v_1\ v_3\ v_4\ v_2\ v_5\ v_6$

glBegin() – glEnd()

- Allow commands inside glBegin()

```
glVertex*();
glColor*();
glIndex*();
glNormal*();
glTextCoord*():
glMultiTextCoord*ARB();
glEdgeFlag*()
glMaterial*();
glArrayElement();
glEvalCoord*();
glEvalPoint*();
glCallList();
glCallLists();
```

Part 2
Computer Graphics - Continuation

References

- Mouse
  - B. Frohlich, J. Plate, J. Wind, G. Wesche, M. Gobel. Cubic-Mouse-based interaction in virtual environments. *Computer Graphics and Applications, IEEE*, 20(4):12-15, 2000.
  - S. Zhai. User performance in relation to 3D input device design. *ACM SIGGRAPH Computer Graphics*, 32(4):50-54, 1998.
  - C. Ware, D. R. Jessome. Using the bat: a six-dimensional mouse for object placement. *Computer Graphics and Applications, IEEE*, 8(6):65-70, 1988.
- OpenGL
  - R. Wolfe. Open GL: agent of change or sign of the times? *ACM SIGGRAPH Computer Graphics*, 32(4):29-31, 1998.

Event Control

- GLUT
  - glutKeyBoardFunc(void (**func*) (unsigned char *key*, int *x*, int *y*))
    - Link keyboard and mouse to an event
  - glutMouseFunc(void(**func*) (int *button*, int *state*, int *x*, int *y*))
    - Link keyboard and mouse to an event
  - glutMotionFunc(void(**func*) (int *x*, int *y*))
    - Routine that is call back when mouse is moved while a mouse button is also pressed
  - void glutSpecialFunc(void (**func*)(int **key**, int *x*, int *y*));
    - Sets the special keyboard callbacks: GLUT_KEY_F1,…, GLUT_KEY_LEFT,…, GLUT_PAGE_UP, GLUT_PAGE_DOWN, GLUT_HOME

Simple Rotation Model

```
// Rotation amounts
static GLfloat xRot = 0.0;
static GLfloat yRot = 0.0;
////////////////////////////////////////////
// main function
int main(int argc, char* argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE |
                       GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(800,600);
   glutCreateWindow("RGB Cube");
   glutReshapeFunc(ChangeSize);
   ////////////////////////////////////////
   // rotation using keys
   glutSpecialFunc(SpecialKeys);
   ////////////////////////////////////////
   // Drawing scene
   glutDisplayFunc(RenderScene);
   SetupRC();
   glutMainLoop();
   return 0;
}
```

```
////////////////////////////////////////////
// Get Arrow Keys
void SpecialKeys(int key, int x, int y)
{
   if(key == GLUT_KEY_UP)
      xRot - = 5.0;
   if(key == GLUT_KEY_DOWN)
      xRot += 5.0;
   if(key == GLUT_KEY_LEFT)
      yRot - = 5.0;
   if(key == GLUT_KEY_RIGHT)
      yRot += 5.0;

   if(key > 356.0)
      xRot = 0.0;
   if(key < -1.0)
      xRot = 355.0;
   if(key > 356.0)
      yRot = 0.0;
   if(key < -1.0)
      yRot = 355.0;
   // Refresh the Window
   glutPostRedisplay();
}
```

More Complex Rotation Model

```
/* Rotation amounts */
static int oldX = 0.0;
static int oldY = 0.0;
/* main function */
int main(int argc,char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB |
      GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(winW,winH);
  glutCreateWindow(APP_NAME);
  glutKeyboardFunc(myKeyboard);
  glutDisplayFunc(myDisplay);
  glutReshapeFunc(myResize);
  glutMotionFunc(myMotion);
  glutMouseFunc(myMouse);
  glutIdleFunc(myIdle);
  /* call the initialization function */
  myInit ();
  /* infinite loop */
  glutMainLoop();
  return 0;
}
```

```
/* Called when the mouse moves in our app area */
void myMotion(int x, int y)
{
  if(mState == DOWN){
    gRot[0] - = (float) ((oldY - y) * 180.0) / 100.0;
    gRot[1] - = (float) ((oldX - x) * 180.0) / 100.0;
    clamp(gRot);
    glutPostRedisplay();
  }
  oldX = x;
  oldY = y;
  return;
}

/* Called when the mouse is clicked */
void myMouse(int button, int state, int x, int y)
{
  if(state == GLUT_DOWN){
    switch(button)
    {
      case GLUT_LEFT_BUTTON:
        break;
      case GLUT_RIGHT_BUTTON:
        mState = DOWN;
        oldX = x;
        oldY = y;
        break;
    }
  }else if (state == GLUT_UP)
    mState = UP;
  return;
}
```

Color Perception

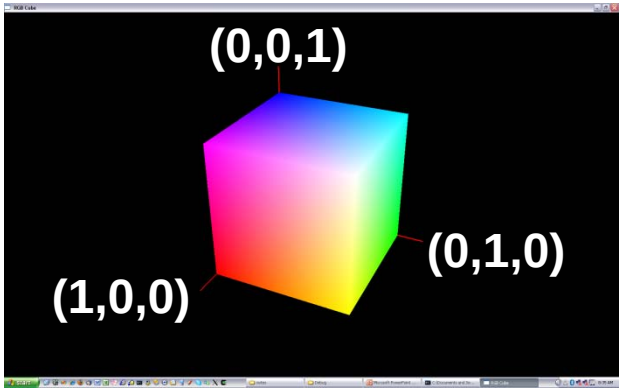
- Visible spectrum
  - 390 nm (violet) – 720 nm (red)
- Rainbow - Spectrum:
  - Violet, indigo, blue, green, yellow, orange, red
- Light sources: distribution of photon frequencies they emit
- White: equal amount of light of all frequencies
- Cones: red – green – blue
- Photons excite cones
  - Different level of excitation of the cones

Monitor

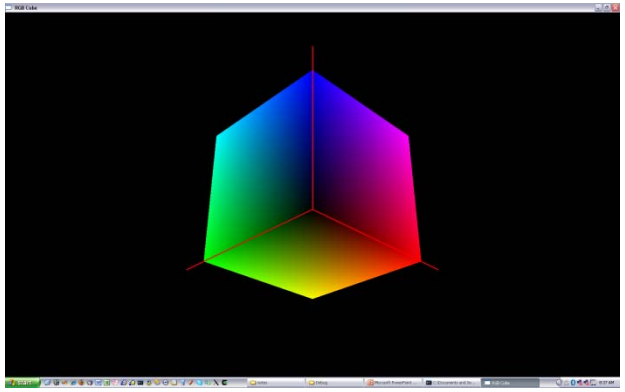
- Pixels: red – green – blue
- Color representation
  - RGB: red – green – blue
  - Different proportions of R, G, and B to stimulate the eye
- Other color representations:
  - HLS: hue – lightness – saturation
  - HSV: hue – saturation – value
  - CMY: cyan – magenta – yellow

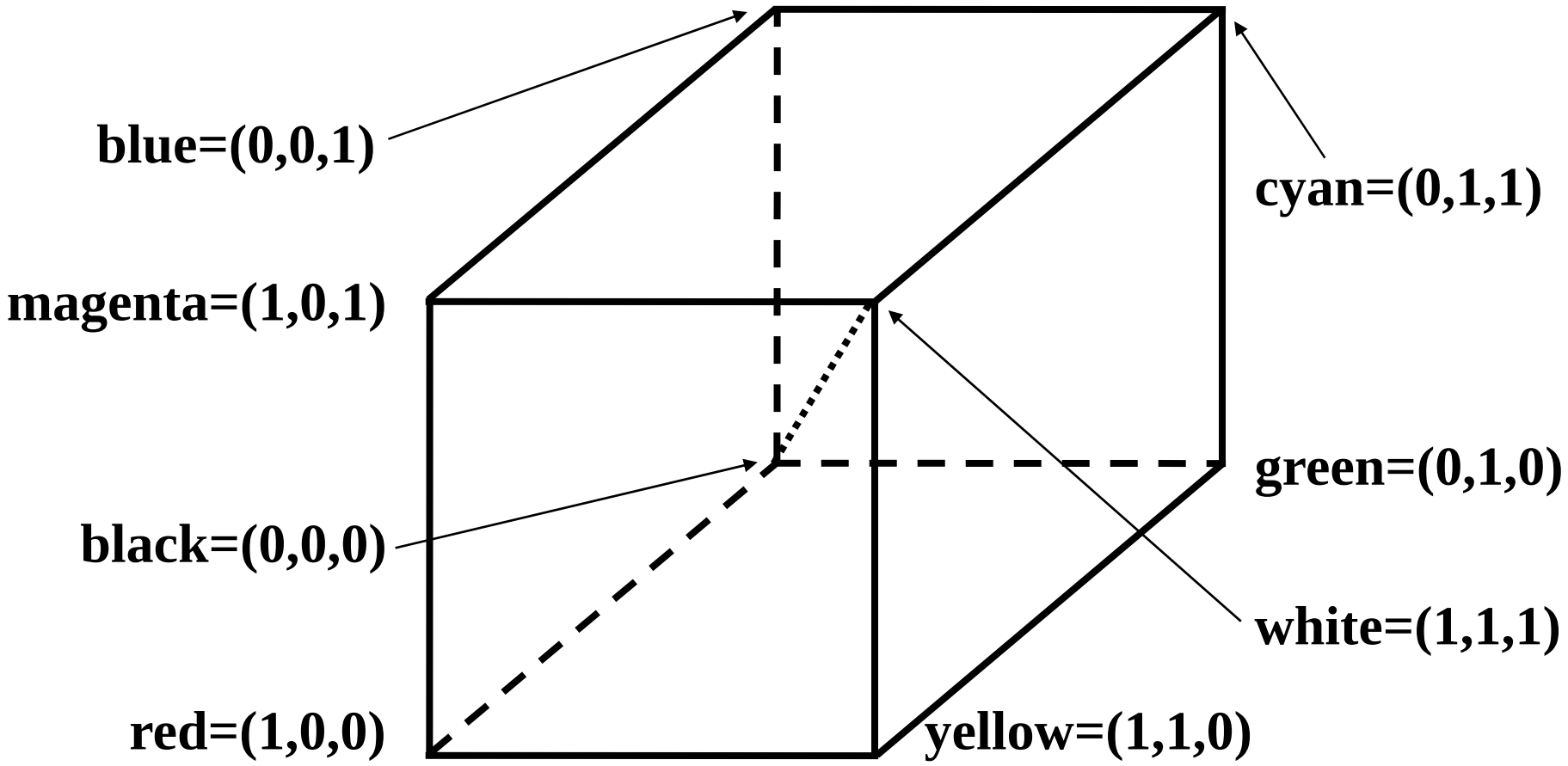
Color Representations

- Hardware oriented
  - RGB
  - CMY
- User oriented:
  - HLS
  - HSV


(0,0,1)
(0,1,0)
(1,0,0)

RGB




blue=(0,0,1)
cyan=(0,1,1)
magenta=(1,0,1)
green=(0,1,0)
black=(0,0,0)
white=(1,1,1)
red=(1,0,0)
yellow=(1,1,0)

CYM

- Cyan, magenta, and yellow
  - complements of red, green, and blue

$$\begin{bmatrix} C \\ Y \\ M \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

CYMK

- Cyan, magenta, yellow, and black (K)

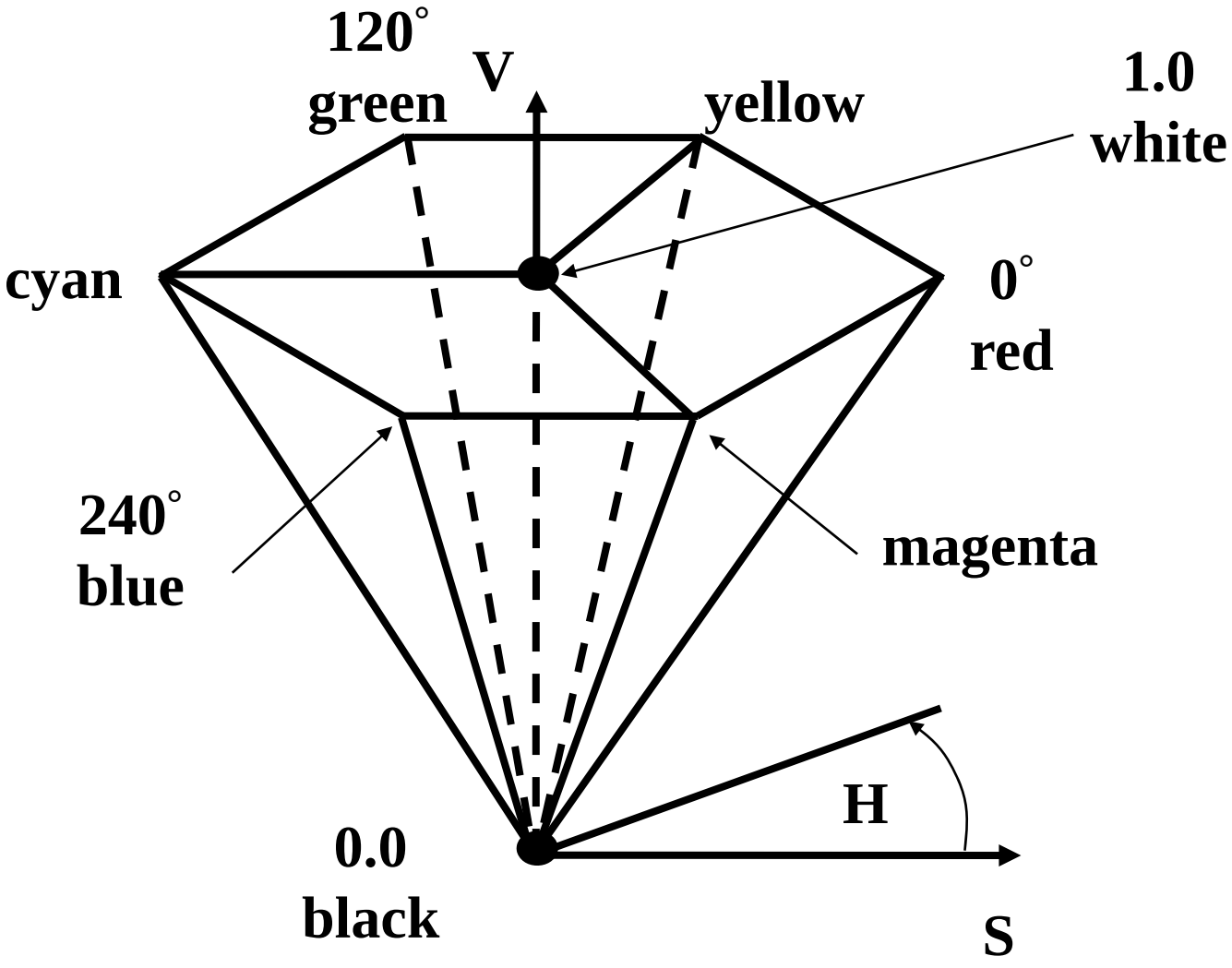
$$K = \min(C, M, Y)$$

$$C = C - K$$

$$M = M - K$$

$$Y = Y - K$$

Single-hexcone HSV



Color in OpenGL: glColor*()

- void glColor3{b s i f d ub us ui}(TYPE *r*,TYPE *g*, TYPE *b*);
- void glColor4{b s i f d ub us ui}(TYPE *r*,TYPE *g*, TYPE *b*, TYPE *alpha*);
- void glColor3{b s i f d ub us ui}v(const TYPE **v*);
- void glColor4{b s i f d ub us ui}v(const TYPE **v*);
- If not alpha value is set, the default is 1.0

Conversions to Floating Point

- **b** **1-byte integer**
  [-128 ; 127] → [-1.0,1.0]
- **s** **2-byte integer**
  [-32,768 ; 32,767] → [-1.0,1.0]
- **i** **4-byte integer**
  [-2,147,483,648 ; 2,147,483,647] → [-1.0,1.0]
- **ub** **1-byte integer**
  [0 ; 255] → [0.0,1.0]
- **us** **2-byte integer**
  [0 ; 65,535] → [0.0,1.0]
- **us** **4-byte integer**
  [0 ; 4,294,967,295] → [0.0,1.0]

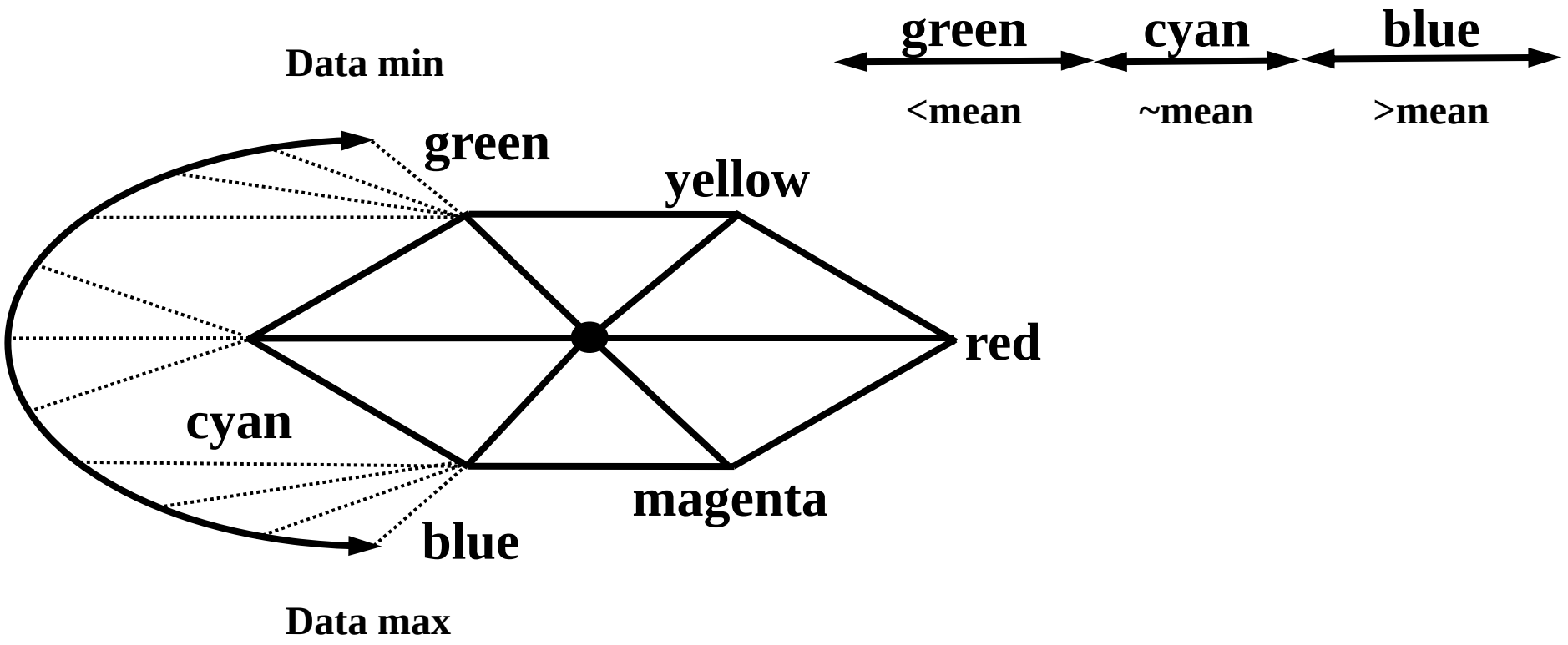
Color in OpenGL: glIndex*()

- void glIndex{s i f d ub}(TYPE *c*);
- void glIndex{s i f d ub}v(const TYPE **c*);
- Sets the current index color
- void glClearIndex (GLfloat *cindex*);

Mapping Data to Colors

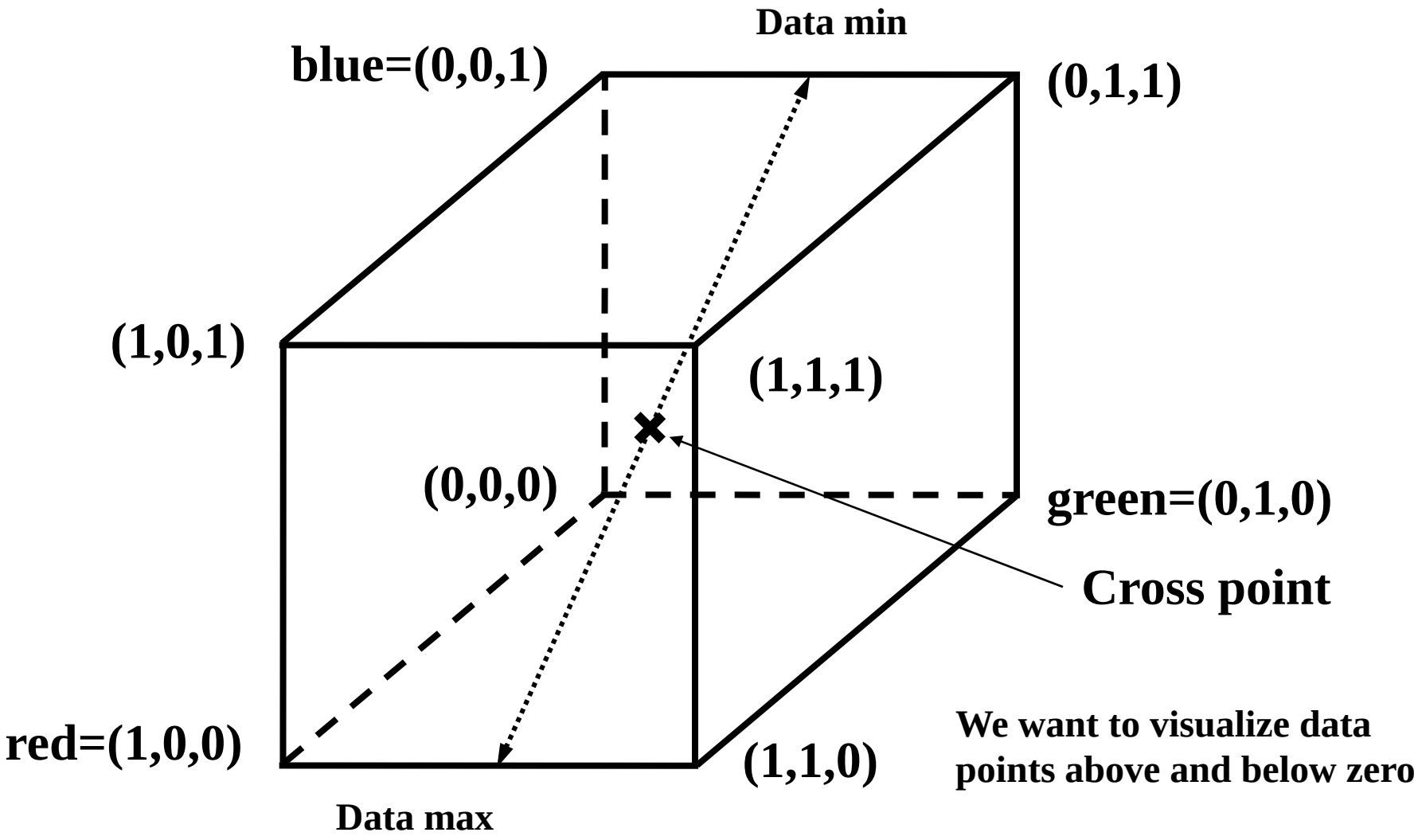
- Find a line or curve in the color space of our choice (RGB, HSV, etc)
- The data values are distributed from minimum to maximum value in the line or curve
- The distribution can be linear or non-linear
  - Linear: equal distance in the line is equal increments in the color map
  - Non-linear: good to highlight ranges of interest in the data

Non-linear Color Map



We want to visualize data points cluster around a special value

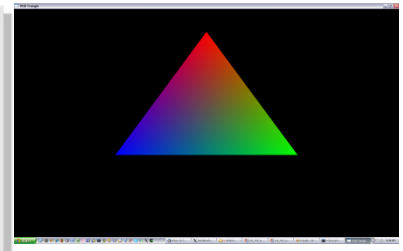
Linear Color Map

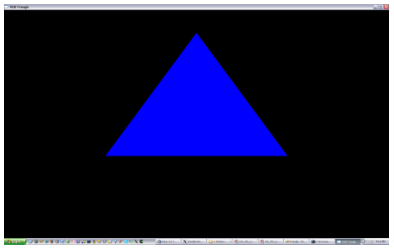


Shading Model

- void glShadowModel(GLenum *mode*);
  - GL_SMOOTH: Many different colors (Gouraud shading)
  - GL_FLAT: One color (default)

```
// Enable smooth shading
glShadeModel(GL_SMOOTH);

// Draw the triangle
glBegin(GL_TRIANGLES);
  // Red Apex
  glColor3f(1.0,0.0,0.0);
  glVertex3f(0.0,200.0,0.0);
  // Green on the right bottom corner
  glColor3f(0.0,1.0,0.0);
  glVertex3f(200.0,-70.0,0.0);
  // Blue on the left bottom corner
  glColor3f(0.0,0.0,1.0);
  glVertex3f(-200.0, -70.0, 0.0);
glEnd();
```





```
// Desable smooth shading
glShadeModel(GL_FLAT);

// Draw the triangle
glBegin(GL_TRIANGLES);
  // Red Apex
  glColor3f(1.0,0.0,0.0);
  glVertex3f(0.0,200.0,0.0);
  // Green on the right bottom corner
  glColor3f(0.0,1.0,0.0);
  glVertex3f(200.0,-70.0,0.0);
  // Blue on the left bottom corner
  glColor3f(0.0,0.0,1.0);
  glVertex3f(-200.0, -70.0, 0.0);
glEnd();
```

Depth Buffer

- z-buffer:
  - draws what is closer to the viewer in the z monitor coordinates
  - If z-buffer is not used the last object drawn is on top

```
// Init section
glutInitDisplayMode(GLUT_DEPTH|...);
glEnable(GL_DEPTH_TEST);

...

// Clear depth buffer
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
// Draw Objects
draw_object_a();
draw_object_a();
```

OpenGL Lighting

- Approximate model
- Light comes from several light sources
  - We can turn on and off sources
- Ambient: light without a particular direction
- Diffuse: light from one direction, it bounces off surfaces in all directions
- Specular: light that bounces off the surfaces and scatters in a preferred direction
- Emissive: light that originates from an object

Create a Light Source

- void glLight{if}(GLenum *light*, GLenum *pname*, TYPE *param*);

- void glLight{if}v(GLenum *light*, GLenum *pname*, TYPE **param*);

- *light:* GL_LIGHT0,…,or GL_LIGHT7
- *pname:* parameter name
- *param:* values to *pname*

- glEnable(GL_LIGHTING);
- glEnable(GL_LIGHT0);

Light Parameters

PNAME	Default Values	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) (0.0, 0.0, 0.0, 1.0)	diffuse intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) (0.0, 0.0, 0.0, 1.0)	specular intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x,y,z,w) position of light
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(x,y,z) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent
GL_SPOT_CUTOFF	180.0	spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor

Attenuation

- The intensity of light decreases as distance from the light source increases:

$$attenuation\_factor = \frac{1}{k_c + k_l d + k_q d^2}$$

$k_c = \mathbf{GL\_CONSTANT\_ATTENUATION}$

$d = \mathbf{distance\ between\ the\ light's\ position\ and\ the\ vertex}$

$k_c = \mathbf{GL\_CONSTANT\_ATTENUATION}$

$k_l = \mathbf{GL\_LINEAR\_ATTENUATION}$

$k_q = \mathbf{GL\_QUADRATIC\_ATTENUATION}$

Lighting Model

- void glLightModel{if}(GLenum *pname*, TYPE *param*);
- void glLightModel{if}v(GLenum *pname*, TYPE **param*);
  - *pname:* parameter name
  - *param:* values to *pname*

PNAME	Default Vaues	Meaning
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient RGBA intensity of light the entire scene
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 or GL_FALSE	how specular reflection angles are computed
GL_LIGHT_MODEL_TWO_SIDE	0.0 or GL_FALSE	choose between one-sided or two-sided lighting
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	whether specular color is calculated separately from ambient and diffuse

OpenGL Materials

- They have different:
  - Ambient colors
  - Diffuse colors
  - Specular colors
- Color components
  - Reflected proportions of the colors
  - R=1.0, G=0.5, B=0.0
    - The material reflects:
      - 100% of read, 50% of green and nothing of blue

Defining Material Properties

- void glMaterial{if}(GLenum *face*, GLenum *pname*, TYPE *param*);

- void glMaterial{if}v(GLenum *face*, GLenum *pname*, TYPE **param*);

- *face:* GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK
- *pname:* parameter name
- *param:* values to *pname*

- glEnable(GL_LIGHTING);
- glEnable(GL_LIGHT0);

Material Parameters

PNAME	Defaults Values	Meaning
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient color of material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffuse color of material
GL_AMBIENT_AND_DIFFUSE		ambient and diffuse color of material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	specular color of material
GL_SHINENESS	0.0	specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissive color of material
GL_COLOR_INDEXES	(0, 1, 1)	ambient, diffuse, and specular color indices

Display Lists

- Lists may improve performance
- Stores commands into a list
- Good when running an application remotely
- It can improve performance when running locally
- Once a list is created is not possible to modify it
- Performance can vary with the OpenGL implementation
- Starting a display list has some overheads

Naming and Creating a Display List

- GLuint glGenLists(GLsizei *range*);
  - Create list
- void glNewList(GLuint *list*, GLenum *mode*);
  - *mode:* GL_COMPILE_AND_EXECUTE, GL_COMPILE
  - Beginning of the list
- void glEndList(void);
  - End of the list
- void glCallList(GLuint *list*);
  - Executes the list

```
// Naming and Creating a List
listIndex = glGenList(1);
If (listIndex != 0){
   glNewList(listIndex, GL_COMPILE);
   …
   glEndList();
}
```

Some Fun Tutorials

- Download Nate Robins' Tutorials
  - http://www.xmission.com/~nate/tutors.html