CPSC/ECE 4780/6780

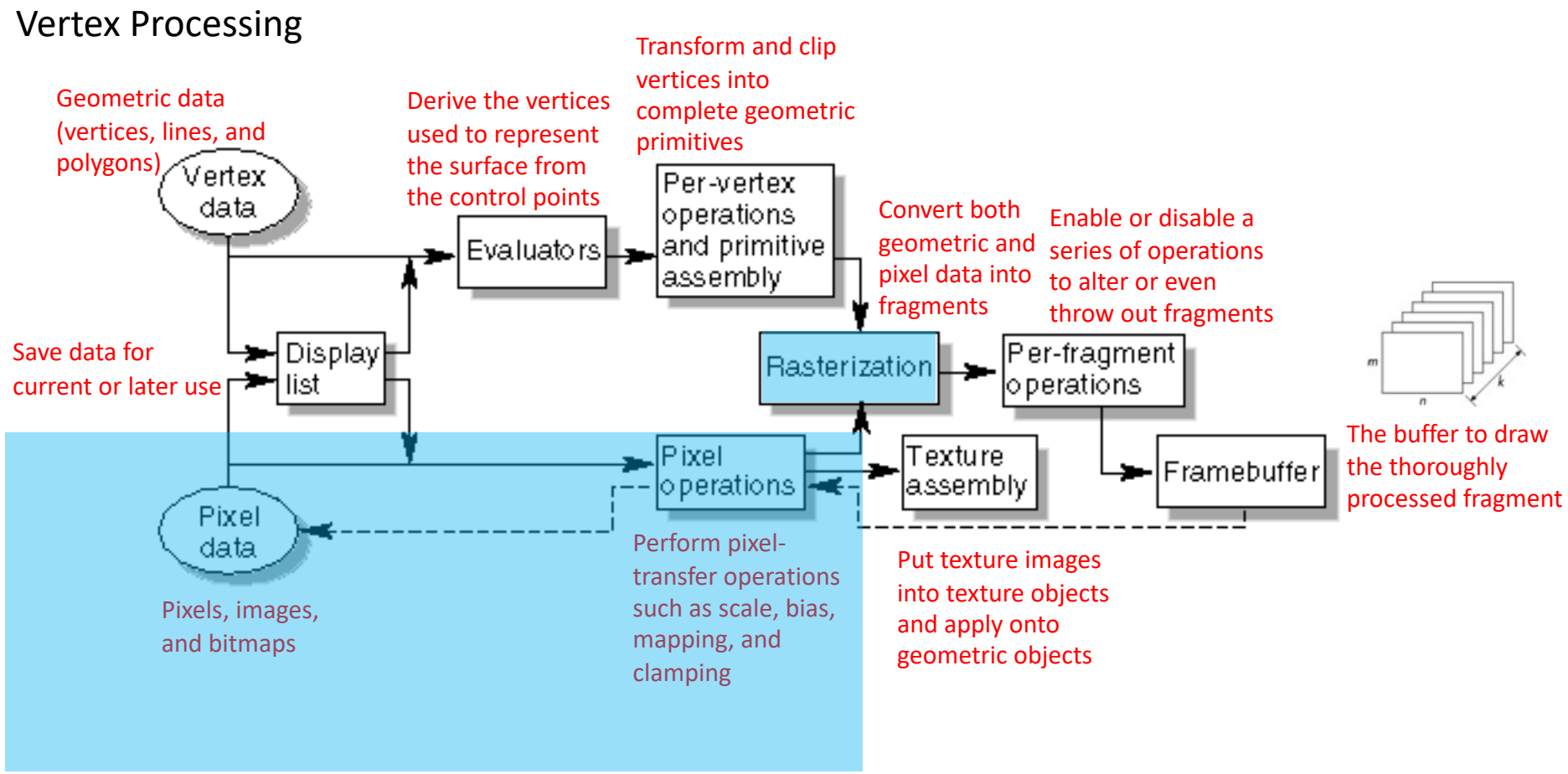
General-Purpose Computation on Graphical Processing Units (GPGPU)

Lecture 19: (OpenGL) Bitmaps, Images, and Pixels

Recap of Last Lecture

- Lights
  - Why develop light models in OpenGL?
  - What is Phong lighting model?
    - How many different types of lightings in a Phong lighting model?
  - How many different light sources?
  - OpenGL lighting function: glLight()
- Materials
  - Define material properties specific to each object
  - glMaterial()
  - glColorMaterial()

Revisit OpenGL Rendering Pipeline



Bitmaps and Images

- Both bitmaps and images take the form of rectangular arrays of pixels
  - Bitmaps
    - Typically used for characters in fonts
    - Consists of a single bit of information about each pixel
    - Like masks in that they're used to overlay another image
  - Image data:
    - Might have been scanned in or calculated
    - Includes several pieces of data per pixel
    - Overwrites or is blended with whatever data is in the framebuffer

Bitmaps

- A bitmap is a rectangular array of 0s and 1s that serves as a drawing mask for a corresponding rectangular portion of the window
- The most common use of bitmaps is for drawing characters on the screen
  - Step 1: Set the current raster position
  - Step 2: Draw the bitmap
  - Step 3: Choose a color for the bitmap

Step 1: Set the Current Raster Position

- The current raster position is the origin where the next bitmap (or image) is to be drawn
- void **glRasterPos**{234}{sifd}(TYPE x, TYPE y, TYPE z, TYPE w);
- void **glRasterPos**{234}{sifd}v(TYPE *coords);
  - Sets the current raster position
  - *x, y, z, w*: specify the coordinates of the raster position
- Specify the raster position in screen coordinates requires specification of the modelview and projection matrices
  - glMatrixMode(GL_PROJECTION);
  - glLoadIdentity();
  - gluOrtho2D(0.0, (GLfloat) width, 0.0, (GLfloat) height);
  - glMatrixMode(GL_MODELVIEW);
  - glLoadIdentity();
- Query the current raster position by glGetFloatv() with argument GL_CURRENT_RASTER_POSITION

Step 2: Draw the Bitmap

- void **glBitmap**(GLsizei width, GLsizei height, GLfloat xbo, GLfloat ybo, GLfloat xbi, GLfloat ybi, const GLubyte *bitmap);
  - Draws the bitmap specified by *bitmap* (a pointer to the bitmap image)
  - *width, height*: the width and height, in pixels, of the bitmap
  - *xbo, ybo*: define the origin of the bitmap
    - Positive values move the origin up and to the right of the raster position
    - Negative values move it down and to the left)
    - Allowing the origin of the bitmap to be placed arbitrarily makes it easy for characters to extend below the origin or to extend beyond the left of the origin
  - *xbi, ybi:* indicate the x and y increments that are added to the raster position after the bitmap is rasterized

Step 3: Choose a Color for the Bitmap

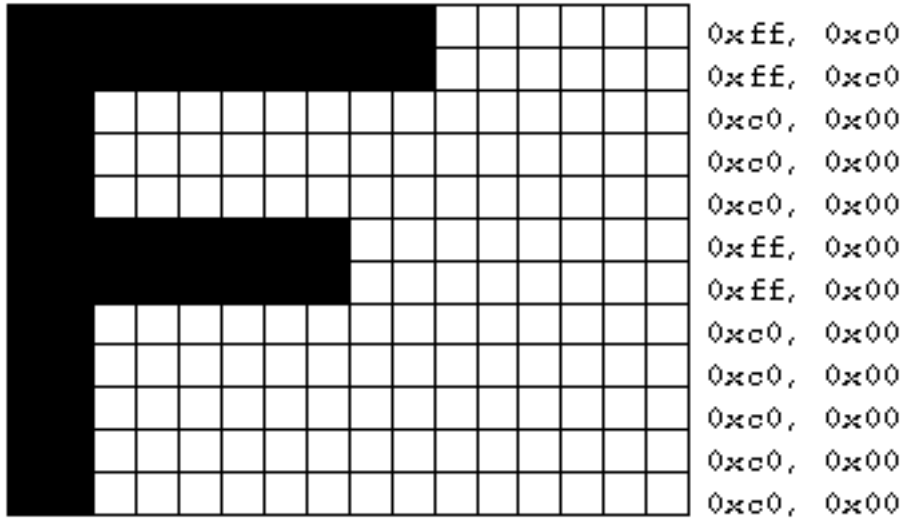
- Set state variables:
  - GL_CURRENT_RASTER_COLOR
  - GL_CURRENT_RASTER_INDEX
- **The raster color state variables are set when glRasterPos*() is called**
- What is the color of the following bitmap? It is white!
  - glColor3f(1.0, 1.0, 1.0); /* white */
  - glRasterPos3fv(position);
  - glColor3f(1.0, 0.0, 0.0); /* red */
  - glBitmap(....);
- Query the current raster color by glGetFloatv() with argument GL_CURRENT_RASTER_COLOR, or glGetIntegerv() with GL_CURRENT_RASTER_INDEX

Fonts and Display-lists

- A font typically consists of a set of characters, where each character has an identifying number (usually the ASCII code) and a drawing method
  - E.g., A => 65, B => 66, …, the string “DAB” would be represented by the three indices 68, 65, 66 (display-lists)
- void **glCallLists**(GLsizei n, GLenum type, const GLvoid *lists);
  - *n*: the number of characters to be drawn
  - *type*: is usually GL_BYTE
  - *lists:* an array of character codes
- To draw character strings in multiple fonts and sizes, the solution is to add an offset to every entry in the string and to choose the display list
  - To set the offset, use the command glListBase()
- GLuint **glGenLists**(GLsizei range);
  - Returns a block of range display-list identifiers
  - The returned lists are all marked as "used" even though they're empty
    - E.g., if glGenLists(4) returns 81, you can use display-list identifiers 81, 82, 83, and 84 for your character
    - If glGenLists() can't find a block of unused identifiers of the requested length, it returns 0

	A	B	C	offset
Font 1:	1065	1066	1067	1000
Font 2:	2065	2066	2067	2000

A Bitmapped Character “F”



Bitmapped F and its data

- The visible part of the F character is at most 10 bits wide
- The bits making up a bitmap are drawn starting from the lower-left corner
- The bitmap is stored in memory in this order - the array of rasters begins with 0xc0, 0x00, 0xc0, 0x00 for the bottom two rows of the F and continues to 0xff, 0xc0, 0xff, 0xc0 for the top two rows.

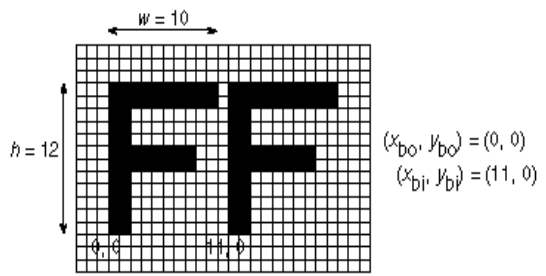
Example: Draw a Bitmapped Character “F”

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

GLubyte rasters[24] = {
   0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
   0xff, 0x00, 0xff, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
   0xff, 0xc0, 0xff, 0xc0};

void init(void)
{
   glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
   glClearColor (0.0, 0.0, 0.0, 0.0);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glRasterPos2i (20, 20);
   glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
   glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
   glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
   glFlush();
}
```




```
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho (0, w, 0, h, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(100, 100);
   glutInitWindowPosition(100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Images

- An image is similar to a bitmap, but contains more information
- Sources of images:
  - A photograph that's digitized with a scanner
  - An image that was first generated on the screen by a graphics program using the graphics hardware and then read back, pixel by pixel
  - A software program that generated the image in memory pixel by pixel
- Images can be used for texture maps
- OpenGL provides three basic commands to manipulate image data:
  - glReadPixels(): frame buffer to processor memory
  - glDrawPixels(): processor memory to frame buffer
  - glCopyPixels(): frame buffer to frame buffer

glReadPixels()

- glReadPixels() - Reads a rectangular array of pixels from the framebuffer and stores the data in processor memory
- void **glReadPixels**(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels);
  - *x, y*: defines lower-left corner of the framebuffer rectangle
  - *width, height*: dimension of the framebuffer rectangle
  - *pixels*: pointer to the array to store pixel data
  - *format*: the kind of pixel data elements that are read (an index value or an R, G, B, or A component value)
  - *type*: the data type of each element
- Call glReadBuffer() to control the current read source buffer

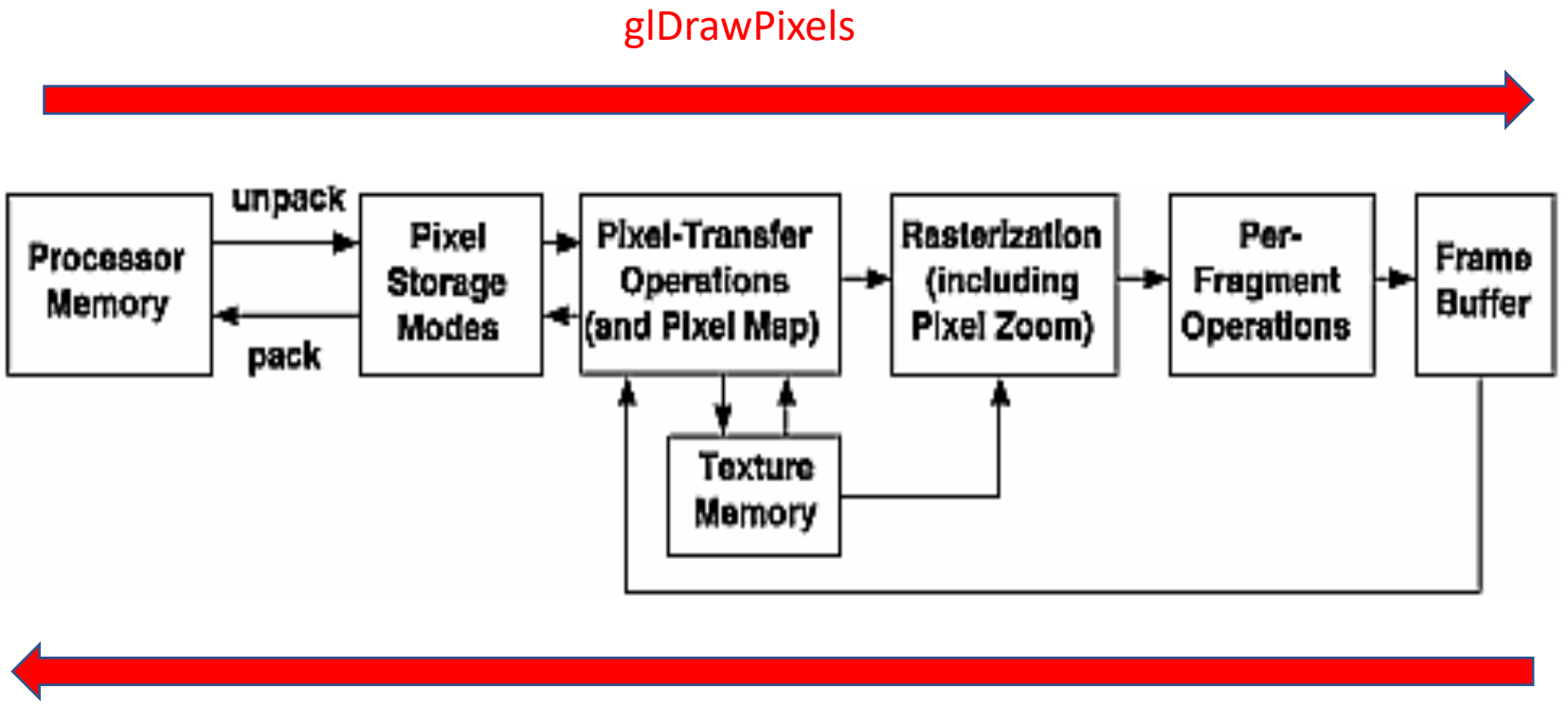
glDrawPixels()

- glDrawPixels() - Writes a rectangular array of pixels from data kept in processor memory into the framebuffer at the current raster position specified by glRasterPos*()
- void **glDrawPixels**(GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels);
  - *width, height*: the dimension of the rectangle of pixel data to be drawn
  - *format*: the kind of pixel data elements that are read (an index value or an R, G, B, or A component value)
  - *type*: the data type of each element
  - *pixels*: pointer to the array that contains the pixel data to be drawn

glCopyPixels()

- glCopyPixels() - Copies a rectangular array of pixels from one part of the framebuffer to another
  - Behaves similarly to a call to glReadPixels() followed by a call to glDrawPixels(), but the data is never written into processor memory
- void **glCopyPixels**(GLint x, GLint y, GLsizei width, GLsizei height, GLenum buffer);
  - *x, y*: defines lower-left corner of the framebuffer rectangle
  - *width, height*: dimension of the framebuffer rectangle
  - *buffer*: the framebuffer being used, either GL_COLOR, GL_STENCIL, or GL_DEPTH
- The read source buffer and the destination buffer of glCopyPixels() are specified by glReadBuffer() and glDrawBuffer() respectively

Imaging Pipeline



Pixel Packing and Unpacking

- Refer to the way that pixel data is written to and read from processor memory
- An image stored in memory has between one and four chunks of data (element), e.g.,
  - Color index: integers
  - Luminance (weighted sum of the RGB values): floating-point values
  - RGBA for each pixel: floating-point values
- Elements can be stored in memory as various data types, ranging from 8-bit bytes to 32-bit integers or floating-point numbers

Controlling Pixel-Storage Modes

- Image data is typically stored in processor memory in rectangular two- or three-dimensional arrays
- All the possible pixel-storage modes are controlled with the glPixelStore*() command
- void **glPixelStore**{if}(GLenum pname, TYPEparam);
  - *Pname*: parameter names such as GL_UNPACK* parameters control how data is unpacked from memory by glDrawPixels(), glBitmap(), etc, and the GL_PACK* parameters control how data is packed into memory by glReadPixels()
  - *Param*: the value that pname is set to

Pixel-Transfer Operations

- Conversions performed during the transfer of pixels to and from the framebuffer are called pixel-transfer operations
- They're controlled with the glPixelTransfer*() and glPixelMap*() commands
  - void **glPixelTransfer**{if}(GLenum pname, TYPEparam);
    - Sets pixel-transfer modes that affect the operation of glDrawPixels(), glReadPixels(), glCopyPixels. E.g.,
      - glPixelTransferf(GL_RED_SCALE, s);
      - glPixelTransferf(GL_GREEN_SCALE, s);
      - glPixelTransferf(GL_BLUE_SCALE, s);
  - void **glPixelMap**{ui us f}v(GLenum map, GLint mapsize, const TYPE *values);
    - Loads the pixel map indicated by *map* with *mapsize* entries, whose values are pointed to by *values*
    - Default sizes are all 1 and the default values are all 0

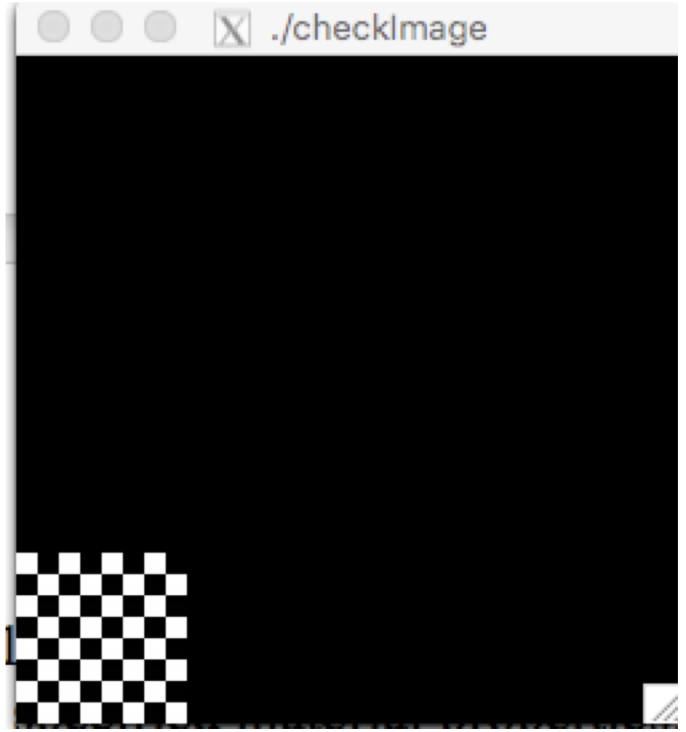
Magnifying, Reducing, or Flipping an Image

- Normally, each pixel in an image is written to a single pixel on the screen
- However, you can arbitrarily magnify, reduce, or even flip (reflect) an image by using glPixelZoom()
- void **glPixelZoom**(GLfloat zoomx, GLfloat zoomy);
  - Sets the magnification or reduction factors for pixel-write operations (glDrawPixels() or glCopyPixels()), in the x- and y-dimensions
  - By default, zoomx and zoomy are 1.0

Example: checkImage.c

```
 1 #include <GL/gl.h>
 2 #include <GL/glu.h>
 3 #include <GL/glut.h>
 4 #include <stdlib.h>
 5 #include <stdio.h>
 6
 7 #define checkImageWidth 64
 8 #define checkImageHeight 64
 9 GLubyte checkImage[checkImageHeight][checkImageWidth][3];
10 static GLdouble zoomFactor = 1.0;
11 static GLint height;
12
13 void makeCheckImage(void)
14 {
15   int i, j, c;
16   for (i = 0; i < checkImageHeight; i++) {
17     for (j = 0; j < checkImageWidth; j++) {
18       c = ((((i&0x8)==0)^((j&0x8))==0))*255;
19       checkImage[i][j][0] = (GLubyte) c;
20       checkImage[i][j][1] = (GLubyte) c;
21       checkImage[i][j][2] = (GLubyte) c;
22     }
23   }
24 }
25 void init(void)
26 {
27   glClearColor (0.0, 0.0, 0.0, 0.0);
28   glShadeModel(GL_FLAT);
29   makeCheckImage();
30   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
31 }
32
33 void display(void)
34 {
35   glClear(GL_COLOR_BUFFER_BIT);
36   glRasterPos2i(0, 0);
37   glDrawPixels(checkImageWidth, checkImageHeight, GL_RGB, GL_UNSIGNED_BYTE, checkImage);
38   glFlush();
39 }
```

Creates a 64-by-64 RGB array of a black-and-white checkerboard image



Draw an pixel rectangle in the lower-left corner of a window

Example: checkImage.c

```
41 void reshape(int w, int h)
42 {
43   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
44   height = (GLint) h;
45   glMatrixMode(GL_PROJECTION);
46   glLoadIdentity();
47   gluOrtho2D(0.0, (GLdouble) w, 0.0, (GLdouble) h);
48   glMatrixMode(GL_MODELVIEW);
49   glLoadIdentity();
50 }
51
52 void motion(int x, int y)
53 {
54   static GLint screeny;
55   screeny = height - (GLint) y;
56   glRasterPos2i (x, screeny);
57   glPixelZoom (zoomFactor, zoomFactor);
58   glCopyPixels (0, 0, checkImageWidth, checkImageHeight, GL_COLOR);
59   glPixelZoom (1.0, 1.0);
60   glFlush ();
61 }
62
63 void keyboard(unsigned char key, int x, int y)
64 {
65   switch (key) {
66     case 'r':
67     case 'R':
68       zoomFactor = 1.0;
69       glutPostRedisplay();
70       printf ("zoomFactor reset to 1.0\n");
71       break;
72     case 'z':
73       zoomFactor += 0.5;
74       if (zoomFactor >= 3.0)
75         zoomFactor = 3.0;
76       printf ("zoomFactor is now %4.1f\n", zoomFactor);
77       break;
78     case 'Z':
79       zoomFactor -= 0.5;
80       if (zoomFactor <= 0.5)
81         zoomFactor = 0.5;
82       printf ("zoomFactor is now %4.1f\n", zoomFactor);
83       break;
84     case 27:
85       exit(0);
86       break;
87     default:
88       break;
89   }
90 }
91
92 int main(int argc, char** argv)
93 {
94   glutInit(&argc, argv);
95   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
96   glutInitWindowSize(250, 250);
97   glutInitWindowPosition(100, 100);
98   glutCreateWindow(argv[0]);
99   init();
100  glutDisplayFunc(display);
101  glutReshapeFunc(reshape);
102  glutKeyboardFunc(keyboard);
103  glutMotionFunc(motion);
104  glutMainLoop();
105  return 0;
106 }
```

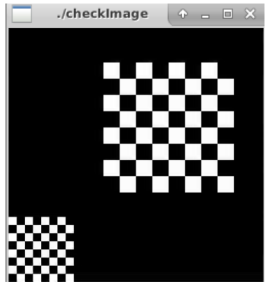


Image Formats

- Images usually are in a standard format (JPEG, TIFF, GIF, PPM)
- PPM: Portable Pixel Map
  - Consists of header followed by all pixel data

    P3
    # comment 1
    # comment 2
    .
    # comment n
    Rows columns max
    pixels

Reading the Header

```
 1 FILE *fd;
 2 char b[70];
 3
 4 printf("enter file name\n");
 5 scanf("%s", b);
 6 fd = fopen(b, "r");
 7
 8 fscanf(fd, "%[^\n]", b);
 9 if (b[0] != 'P' || b[1] != '3') {
10   printf("%s is not a PPM file!\n", b);
11   exit(0);
12 }
13
14 char c;
15 fscanf(fd, "%c", &c);
16 while (c == '#') {
17   fscanf(fd, "%[^\n]", b);
18   printf("%s\n", b);
19   fscanf(fd, "%c", &c);
20 }
21 ungetc(c, fd);
```

Reading the Data

```
23 int n, m, k;
24 fscanf(fd, "%d %d %d", &n, &m, &k);
25
26 GLubyte *image;
27 int nm = n * m;
28 image = malloc(3*sizeof(GLuint)*nm);
29 float s = 255./k;
30 int red, green, blue;
31 for (int i = 0; i < nm; i++) {
32   fscanf(fd, "%d %d %d", &red, &green, &blue);
33   image[3*nm-3*i-3] = red;
34   image[3*nm-3*i-2] = green;
35   image[3*nm-3*i-1] = blue;
36 }
```