Introduction to Open Graphics Libraries: GL, GLU, GLUT, GLEW

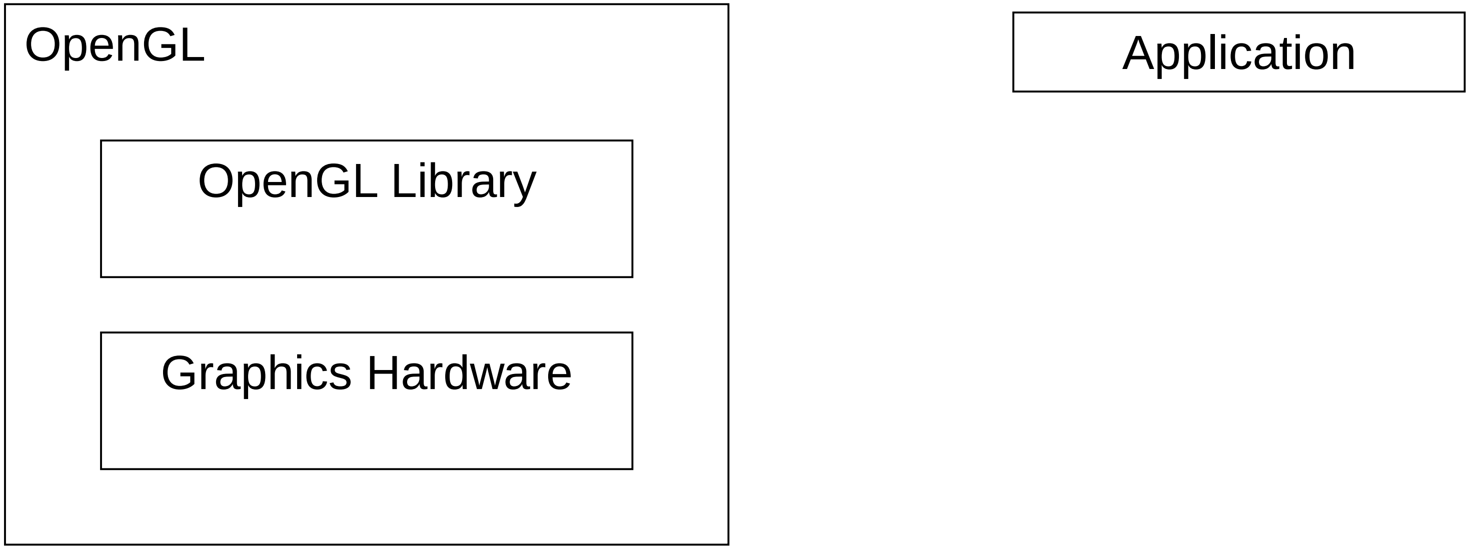
Lecture 2

OpenGL

- OpenGL is a good 3D Graphics API.
  - Allows real time rendering
  - Widely supported
  - Easy to use

What is OpenGL

- GL stands for Graphics Library
  - OpenGL is typically implemented as a library of entry points and graphics hardware to support that library



OpenGL

- In computer systems designed for 3D graphics, the hardware directly supports almost all OpenGL features.
- **OpenGL doesn’t include support for windowing, input, or user interface functionality**, as computer systems typically provide platform-specific support for these features.
- The GLUT library provides platform independent support for this functionality.

Graphics Libraries

- OpenGL (Open Graphics Library)
  - for rendering 2D and 3D computer graphics
  - Version > 2.0 OpenGL Shading Language (GLSL)
    - GLSL is a high-level C-like programming language that allows to write programs in GPU
- GLEW (OpenGL Extension Wrangler)
  - easy to use OpenGL extensions in your programs
- GLUT (freeglut), GLFW (OpenGL FrameWork)
  - for basic windowing functionality

Installation

- Visual Studio 2010
- FreeGLUT
  - http://freeglut.sourceforge.net
- GLEW
  - http://glew.sourceforge.net

freeglut header and library paths

include\GL\freeglut.h
include\GL\glut.h

For Visual Studio 2010:
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include

lib\freeglut.lib
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\lib

bin\freeglut.dll
copy to the directory that contains EXE

GLEW header and library paths

include\GL\glew.h

include\GL\wglew.h

For Visual Studio 2010:

C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include

lib\glew32.lib

C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\lib

bin\glew32.dll

copy to the directory that contains EXE

- **Project | Properties | Linker | Input**
- Under the **Additional Dependencies** section, add glew32.lib; freeglut.lib;

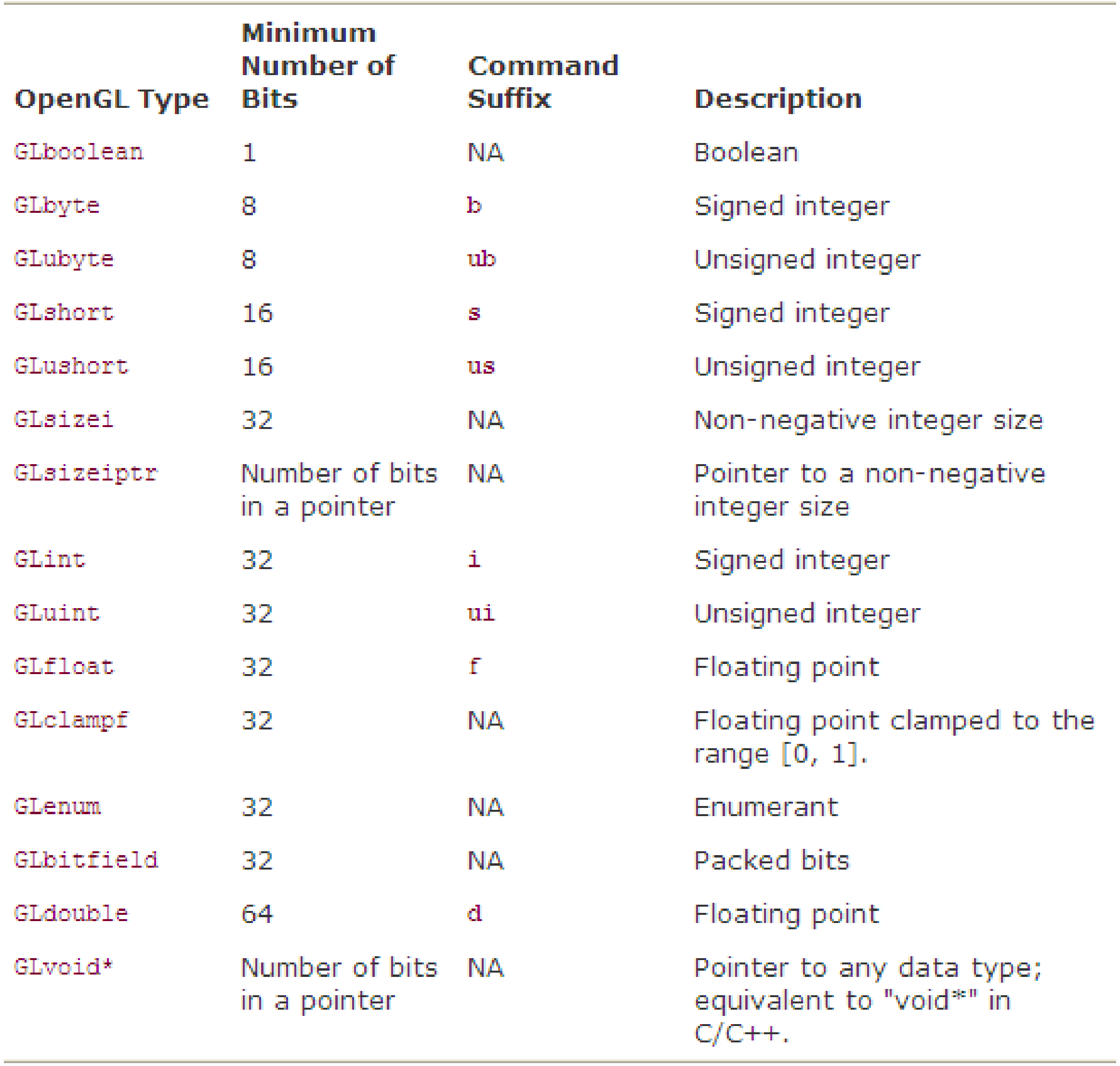
Preliminaries

- Header Files

~~#include <GL/gl.h>~~	(the core library)
#include <GL/glu.h>	(the utility library)
#include <GL/freeglut.h>	(freeware windowing toolkit)
#include <GL/glew.h>	(glsl shading)

- Libraries
- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - GLfloat, GLint, GLenum, etc.

Syntax: OpenGL Data Types



OpenGL Type	Minimum Number of Bits	Command Suffix	Description
`GLboolean`	1	NA	Boolean
`GLbyte`	8	`b`	Signed integer
`GLubyte`	8	`ub`	Unsigned integer
`GLshort`	16	`s`	Signed integer
`GLushort`	16	`us`	Unsigned integer
`GLsizei`	32	NA	Non-negative integer size
`GLsizeiptr`	Number of bits in a pointer	NA	Pointer to a non-negative integer size
`GLint`	32	`i`	Signed integer
`GLuint`	32	`ui`	Unsigned integer
`GLfloat`	32	`f`	Floating point
`GLclampf`	32	NA	Floating point clamped to the range [0, 1].
`GLenum`	32	NA	Enumerant
`GLbitfield`	32	NA	Packed bits
`GLdouble`	64	`d`	Floating point
`GLvoid*`	Number of bits in a pointer	NA	Pointer to any data type; equivalent to "void*" in C/C++.

Commands

- The C-binding implements OpenGL commands as C-callable function prefixed with gl.

More…

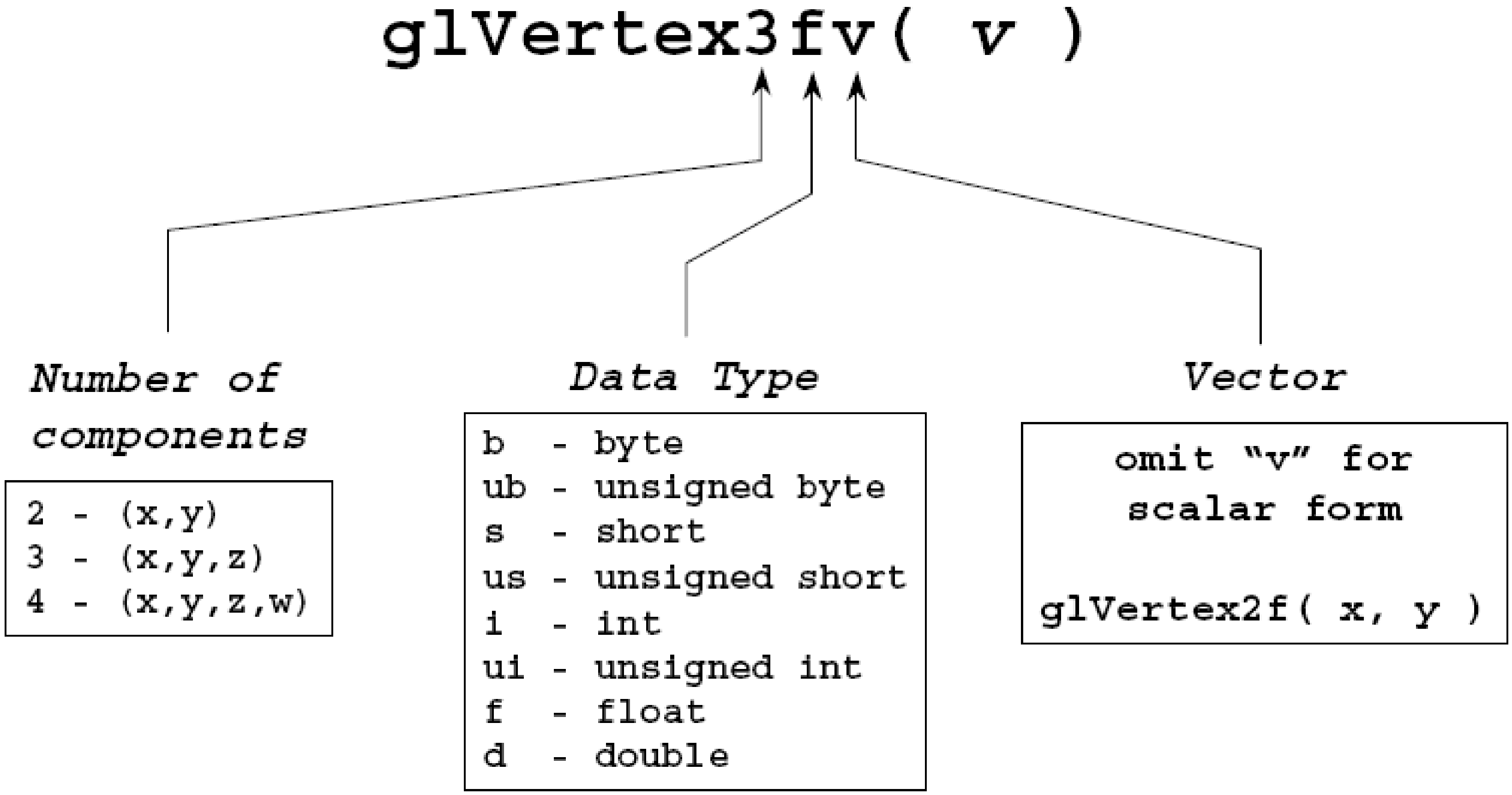
- glColor3f()

  -specifies an RGB color value

- glVertex3f()

  - specifies an xyz vertex location

OpenGL Command Format



`glVertex3fv( v )`

Number of components

2 -	(x,y)
3 -	(x,y,z)
4 -	(x,y,z,w)

Data Type

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

Vector

omit "v" for scalar form

`glVertex2f( x, y )`

OpenGL is a library for rendering computer graphics

- Generally, there are two operations that you do with OpenGL:
  - draw something
  - change the state of how OpenGL draws
- OpenGL has two types of things that it can render: geometric primitives and image primitives.
  - *Geometric primitives* are points, lines and polygons.
  - *Image primitives* are bitmaps and graphics images

OpenGL as a Renderer

- Geometric primitives
  - Points, lines and polygons
- (Raster) Image primitives
  - Images and bitmaps
  - Separate pipeline for images and geometry
    - Linked through texture mapping
  - Rendering depends on state
    - Colors, materials, light sources, etc.

Primitives

Primitives are groups of one or more vertices.

- Point requires single vertex
- Line and filled primitives require two or more vertices.

Vertices have their own color, texture coordinates and normal state.

Specifying Geometric Primitives

- Primitives are specified using

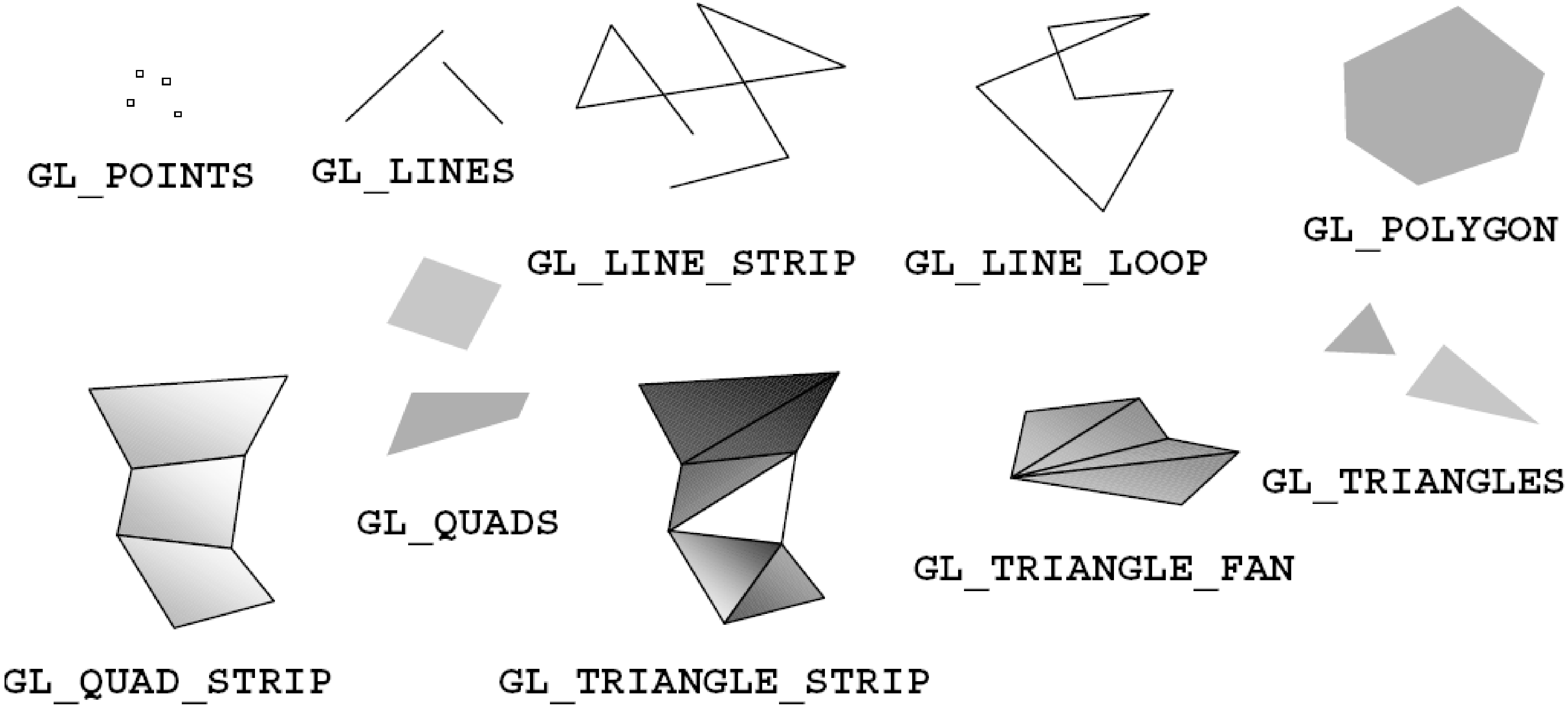
  glBegin(primType);

  glEnd();

  – primType determines how vertices are combined

OpenGL Geometric Primitives

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



Example

```
// Specify an RGB color value with three floats:
GLfloat red=1.f, green=1.f, blue=1.f;
glColor3f( red, green, blue );

// Specify an RGBA color value with four unsigned bytes:
GLubyte r=255, g=255, b=255, a=255;
glColor4ub( r, g, b, a );

// Specify an RGB value with the address of three shorts:
GLshort white[3] = { 32767, 32767, 32767 };
glColor3sv( white );
```

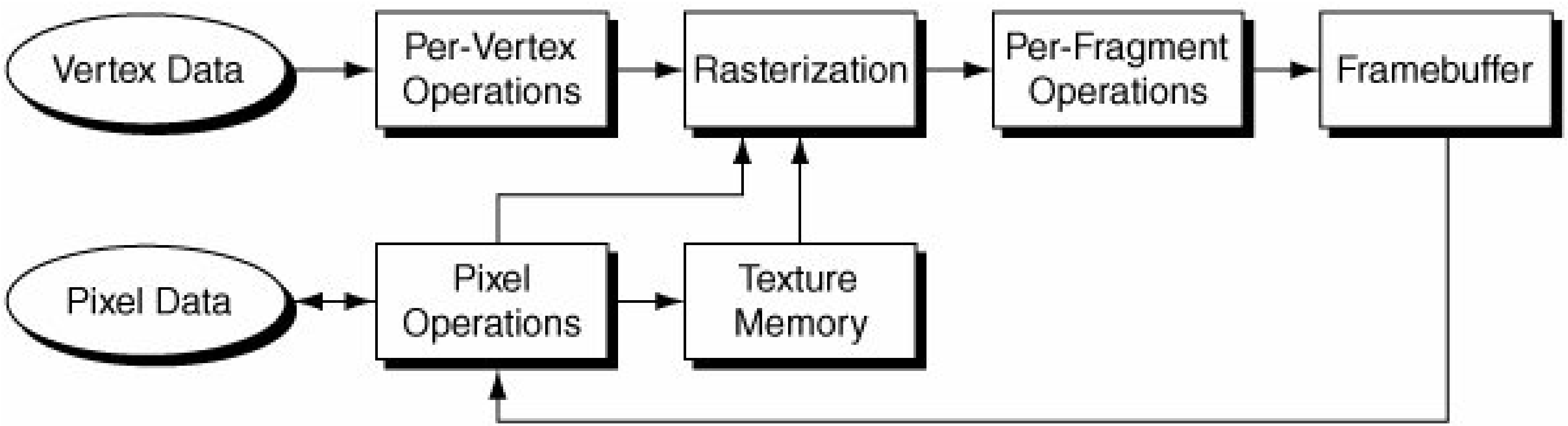
OpenGL is a state machine

```
glColor3f( 1.f, 0.f, 0.f ); // red as an RGB triple
glBegin( GL_POINTS );
  glVertex3f( -.5f, 0.f, 0.f ); // XYZ coordinates of first point
glEnd();

glColor3f( 0.f, 0.f, 1.f ); // blue as an RGB triple
glBegin( GL_POINTS );
  glVertex3f( 0.f, 0.f, 0.f ); // XYZ coordinates of second point
glEnd();

glBegin( GL_POINTS );
  glVertex3f( .5f, 0.f, 0.f ); // XYZ coordinates of third point
glEnd();
```

Rewrite the code…

```
glBegin( GL_POINTS );
  glColor3f( 1.f, 0.f, 0.f ); // red as an RGB triple
  glVertex3f( -.5f, 0.f, 0.f );
           // XYZ coordinates of first point
  glColor3f( 0.f, 0.f, 1.f ); // blue as an RGB triple
  glVertex3f( .5f, 0.f, 0.f );
           // XYZ coordinates of second point
glEnd();
```

The OpenGL pipeline architecture



Per Vertex Operations

- **Transformation**

  OpenGL transforms each vertex from object-coordinate space to window-coordinate space.

- **Lighting**

  If the application has enabled lighting, OpenGL calculates a lighting value at each vertex.

- **Clipping**

  If a primitive is partially visible, OpenGL clips the primitive so that only the visible portion is rasterized.

Pixel Operations

- OpenGL performs pixel storage operations on all blocks of pixel data that applications send to and receive from OpenGL.
- These operations control byte swapping, padding, and offsets into blocks of pixel data to support sending and receiving pixels in a wide variety of formats.

Rasterization

- Rasterization converts geometric data into fragments.
- Fragments are position, color, depth, texture coordinate, and other data that OpenGL processes before eventually writing into the framebuffer.
- Contrast this with pixels, which are the physical locations in framebuffer memory where fragments are stored.

Per-fragment operations

- Pixel ownership
- Scissor test
- Multisample fragment operations
- Alpha test
- Stensil test
- Depth test
- Occlusion query
- Blending
- Dithering
- Logical operation

GLUT Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop

simple.c

```
#include <windows.h>
#include <GL/freeglut.h>
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
void setup(void){
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
```

simple.c

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow(“Simple”);
    glutDisplayFunc(display);
    setup();
    glutMainLoop();
    return 0;
}
```

GLUT Callback functions

- Routine to call when something happens
  - – window resize or redraw
  - – user input
  - – animation
- “Register” callbacks with GLUT
    - **glutDisplayFunc(display);**
    - **glutIdleFunc(idle);**
    - **glutKeyboardFunc(keyboard);**

GLUT supports many different callback actions, including:

- glutDisplayFunc()
  - called when pixels in the window need to be refreshed.
- glutReshapeFunc()
  - called when the window changes size
- glutKeyboardFunc()
  - called when a key is struck on the keyboard
- glutMouseFunc()
  - called when the user presses a mouse button on the mouse
- glutMotionFunc()
  - called when the user moves the mouse while a mouse button is pressed
- glutPassiveMouseFunc()
  - called when the mouse is moved regardless of mouse button state
- glutIdleFunc()
  - a callback function called when nothing else is going on. Very useful for animations.

Rendering Callback

Do all of your drawing here

glutDisplayFunc(display);

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_TRIANGLE_STRIP );
        glVertex3fv( v[0] );
        glVertex3fv( v[1] );
        glVertex3fv( v[2] );
        glVertex3fv( v[3] );
    glEnd();
    glutSwapBuffers();
}
```

Idle callbacks

- Use for animation and continuous update

```
glutIdleFunc( idle );

    void idle( void )
    {
            t += dt;
            glutPostRedisplay();
    }
```

User input callbacks

- Process user input

  glutKeyboardFunc(keyboard);

```
void keyboard( char key, int x, int y )
{
    switch( key ) {
        case ‘q’ : case ‘Q’ :
            exit( EXIT_SUCCESS );
            break;
        case ‘r’ : case ‘R’ :
            rotate = GL_TRUE;
            break;
    }
}
```

User input callbacks

- Process special keys

```
glutSpecialFunc(
        void (*func) (int key, int x, int y) )

glutSpecialFunc(specialkey);

void specialkey( int key, int x, int y ) {
   if (key==GLUT_KEY_F1)
        MessageBeep(-1);
        . . .
}
```

Non-ascii Key Values

- GLUT_KEY_F1
- GLUT_KEY_F12
- GLUT_KEY_LEFT
- GLUT_KEY_RIGHT
- GLUT_KEY_UP
- GLUT_KEY_DOWN
- GLUT_KEY_PAGE_UP
- GLUT_KEY_PAGE_DOWN
- GLUT_KEY_HOME
- GLUT_KEY_END
- GLUT_KEY_INSERT

Mouse callbacks

```
glutMouseFunc(
        void (*func) (int button, int state,
                              int x, int y) )
```

- **button:**
  - GLUT_LEFT_BUTTON
  - GLUT_MIDDLE_BUTTON
  - GLUT_RIGHT_BUTTON
- **state:**
  - GLUT_UP
  - GLUT_DOWN

Mouse callbacks

```
glutMouseFunc(mouse);

void mouse( int button, int state, int x, int y ) {
  if (button == GLUT_LEFT_BUTTON &&
         state == GLUT_DOWN)
         MessageBeep(-1);
         . . .
}
```

Timer callback

```
glutTimerFunc(
        unsigned int msecs,
        (*func) (int value),
        int value
)
```