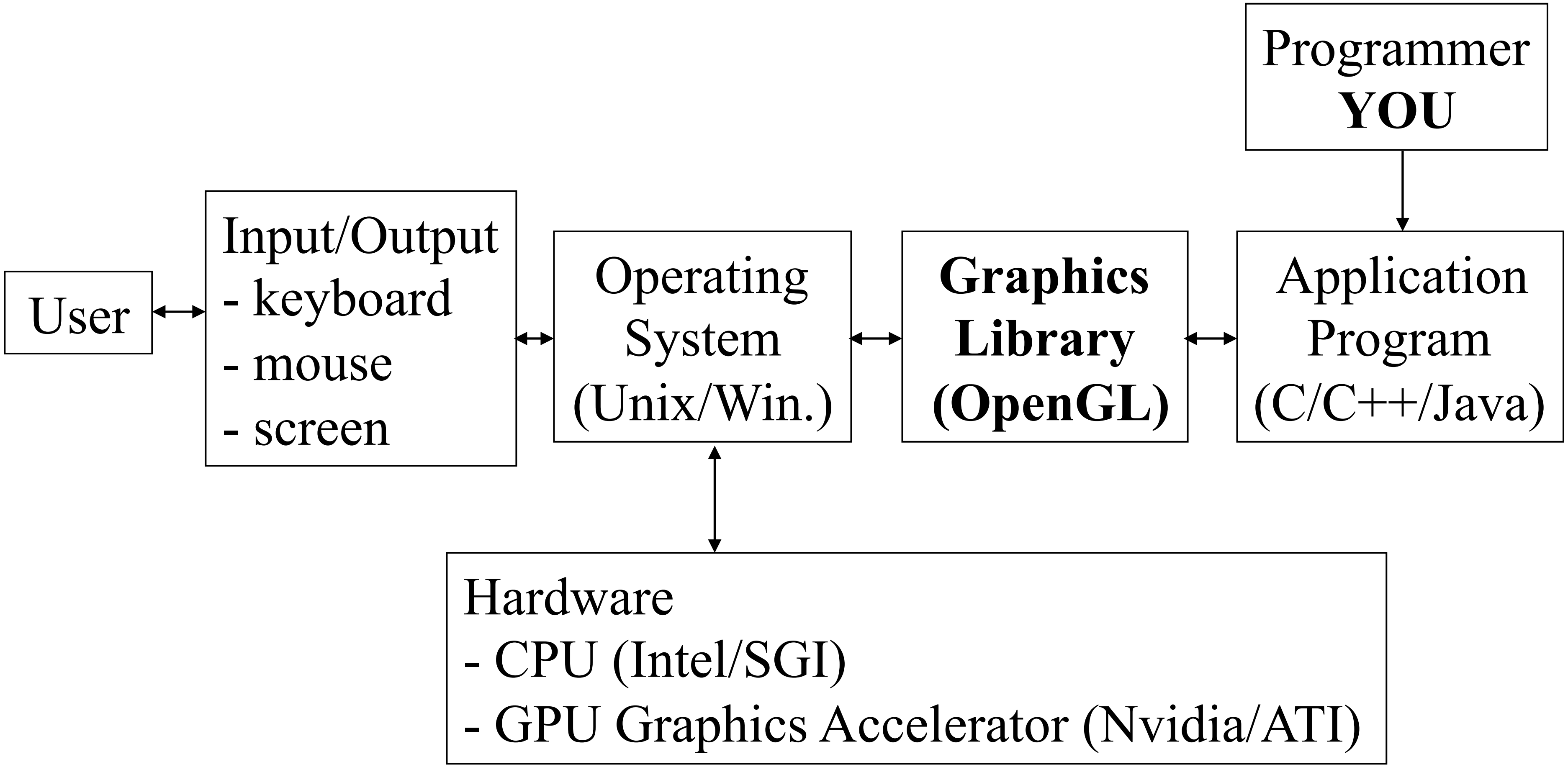
Introduction to OpenGL
“Getting Started”

Reading: Angel Ch.2 or Woo Ch.1

What is OpenGL?

Application programmers interface (API) for 2D/3D graphics



Why OpenGL?

Open standard for graphics based applications

- originally developed by SGI as 'GL' graphics library
- Released as an open-standard
- Widely used for interactive graphics applications
  Animation/VR/Games

Platform independent library of low-level graphics functions

- Approx. 250 distinct commands for 3D graphics
- Hardware accelerated for particular platform
- Very fast 3D rendering

What OpenGL doesn't do:

No functions dependent on a particular platform
No high-level functions for object description etc.
Utility libraries to support platform dependent functions
GLU/GLUT

What OpenGL does & does' nt do

Does:

- Model shape using 3D points/lines/polygons
- Lighting
- Shading
- Texturing of images
- Rendering: clipping/projection/visibility

Doesn' t:

- Limited support for: mirrors, shadows, inter-reflection,
  curved surfaces, motion blur
- Scene hierarchies (OpenSG/VRML/Java-3D)
- User interface functions (X/Windows…)
- Input (mouse/keyboard)

OpenGL API

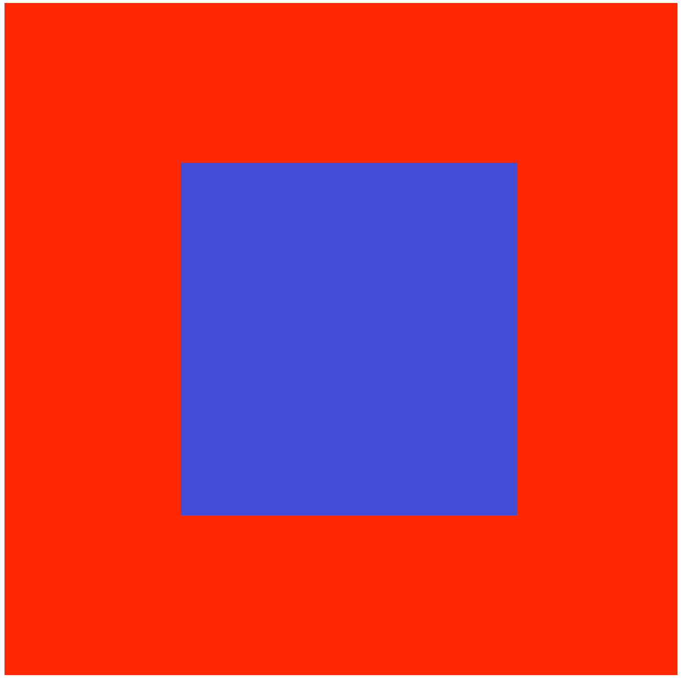
Application Interface for 2D/3D Graphics

- Based on synthetic camera model
- Graphics pipeline:
  3D model - transform - clip - project - rasterise - 2d image
- Library of C-functions to specify:
  Objects
  Viewer
  Lights
  Material Properties
- State machine:
  behaviour determined by a set of global state variables

A Simple Example OpenGL Program: Square

```
#include "gl/gl.h"   /* include functions from gl library  */
main() {
        /* call my function to initialise a draw window here */

        /* OpenGL code to draw a square */
        glClearColor(1.0,0.0,0.0,0.0);               /* set window to red (r,g,b,a) */
        glClear(GL_COLOR_BUFFER_BIT);      /* clear window */

        glOrtho(0.0,1.0, 0.0,1.0,-1.0,1.0);          /* setup 3d coordinate space */

        glColor3f(0.0,0.0,1.0);                      /* set drawing colour blue (r,g,b) */
        glBegin(GL_POLYGON);                         /* specify a polygon */
            glVertex3f(0.25,0.25,0.0)                    /* vertex 1  (x,y,z) */
            glVertex3f(0.75,0.25,0.0)                    /* vertex 2  (x,y,z) */
            glVertex3f(0.75,0.75,0.0)                    /* vertex 3  (x,y,z) */
            glVertex3f(0.25,0.75,0.0)                    /* vertex 4  (x,y,z) */
        glEnd();

        glFlush();   /* draw all objects */

        /* call myfunction to update window and handle events */
}
```

Result of Simple Example Code



OpenGL Syntax

All OpenGL commands have the prefix **'gl'**

glClear()
glColor3f()
glVertex3f()

Constants are defined with prefix **'GL'** & use '_' to separate words

GL_COLOR_BUFFER_BIT

American spelling: Color

OpenGL Variable Types

Type information is appended to the end of the command

glColor**3f**(r,g,b) - a colour of 3 floating point components

glVertex**3f**(x,y,z) - a vertex with 3 floating point coordinates

glVertex**2f**(x,y) - a vertex with 2 floating point coordinates

Different versions of the same function exist for different types

glVertex2i(p,q) - vertex with 2 integer coordinates

Suffix	**Type**	**OpenGL Type**	**C type**
b	8-bit integer	GLbyte	short
i	32-bit integer	GLint	int or long
f	32-bit real	GLfloat	float
d	64-bit real	GLdouble	double
ui	32-bit unsigned int	GLuint	unsigned int

+ others

Use OpenGL Types to avoid problems

OpenGL Arrays or Vectors

Many commands support arrays:

```
GLfloat color_array[] = {1.0,0.0,0.0};   /* rgb array */
glColor3fv(color_array);

GLint coordinate_array[] = {1,7};
glVertex2iv(coordinate_array);
```

To refer to a command which takes multiple types we use ‘*’:

```
glColor*()
glVertex*()
```

One additional type: GLvoid - used for functions that use arrays

OpenGL as a State Machine

OpenGL is a state machine with state variables which control all aspects of modelling/viewing/lighting:

- draw colour
- background colour
- line width
- shading
- antialiasing on/off
- texture on/off
- coordinate system

…..

All state variables have default values and can be changed:

```
glColor3f(1.0,0.0,0.0);          /* set draw colour state to red */
glLineWidth(2.0);                /* set line width state */
glEnable(GL_LINE_STIPPLE);   /* set draw dashed lines */
```

Current 'state' is applied for all subsequent drawing commands

OpenGL Modelling

Primitives: points, lines, polygons (triangle, quadrilateral, n-gon)
+ sets of primitives

Small set of primitives to allow maximum portability

Complex shapes specified by many primitives

OpenGL primitives specified by a list of points:

```
glBegin(type);                /* type is point/line/polygon */
        glVertex*();
        glVertex*();
        glVertex*();
        …..
glEnd();
```

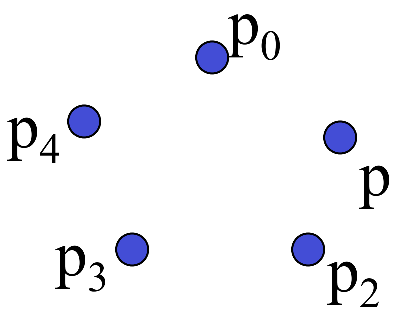
Objects: Utility library GLU contains pre-defined derived objects:
sphere, cylinder ….

Points & Lines

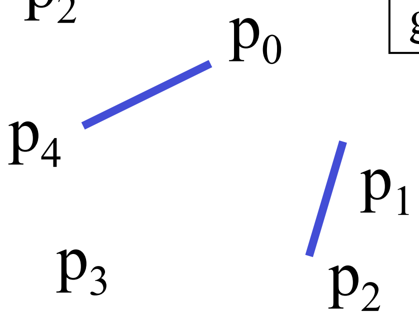
```
glColor3f(0.0,0.0,1.0);
glBegin(type);
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
    glVertex2i(x3,y3);
    glVertex2i(x4,y4);
    glVertex2i(x5,y5);
glEnd();
```

type in glBegin():

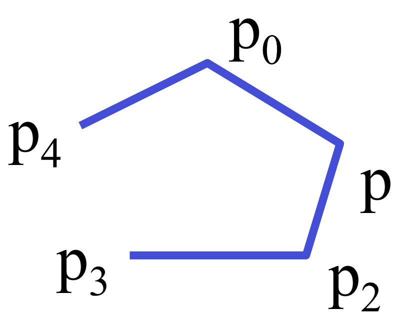
GL_POINTS



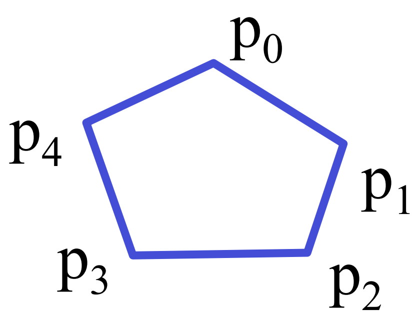
GL_LINES



GL_LINE_STRIP



GL_LINE_LOOP



Convention: Points are numbered from zero $p_0 \ldots p_{n-1}$

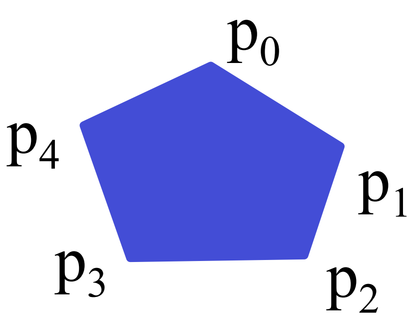
Polygons

Must be: **'Flat'** All vertices lie in a plane
'Simple' Polygon edges do not intersect
'Convex' All point are on one side of any edge

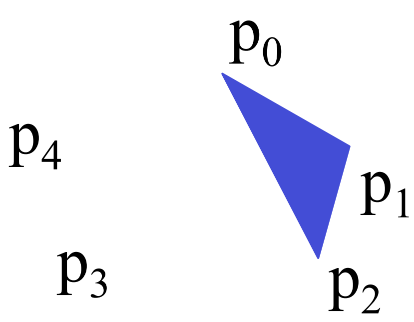
Allows for fast polygon rendering algorithm implement in hardware

Type for glBegin():

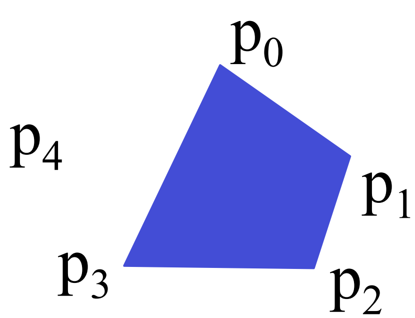
GL_POLYGON



GL_TRIANGLES



GL_QUADS



Convention: Polygons are specified in anticlockwise vertex order

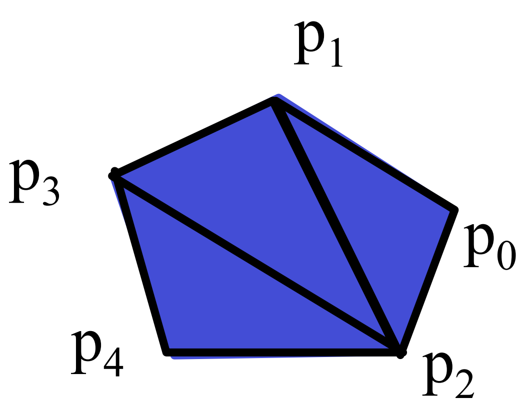
Sets of Polygons

Groups of Triangles or Quadrilaterals that share verticies

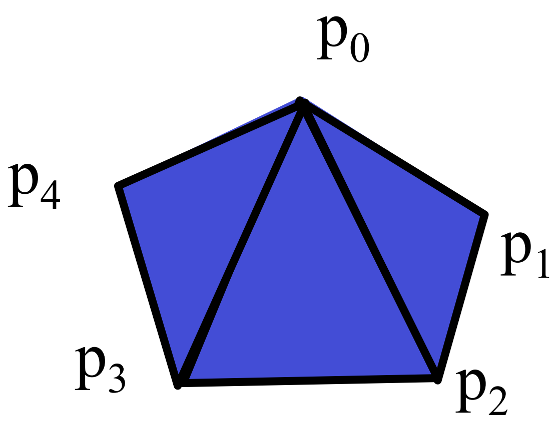
Efficient representation & rendering

Type for glBegin();

GL_TRIANGLE_STRIP GL_TRIANGLE_FAN



Triangles (p_0,p_1,p_2)
(p_1,p_3,p_2)
(p_2,p_3,p_4)



All triangle from p_0

Color in OpenGL

It's just not "colour"!

RGB Three-component additive colour model: red + green + blue

Analogous to human colour perception: 3 colour receptors
Assumption: Any 2 colours are the same if they have the same rgb
(does not allow for distribution of wavelengths)

OpenGL colour components are in the range [0.0,1.0]
- each component represents the intensity of that colour

```
glColor3f(0.1,0.4,0.7);    /* r,g,b colour intensities */
```

Alpha channel - represents the opacity or transparency
- **RGBA** colour model

```
glColor4f(1.0,0.0,0.0,0.5);  /* red semi-transparent */
```

Slide showing RGBA transparency

Viewing in OpenGL

Specification of the camera:

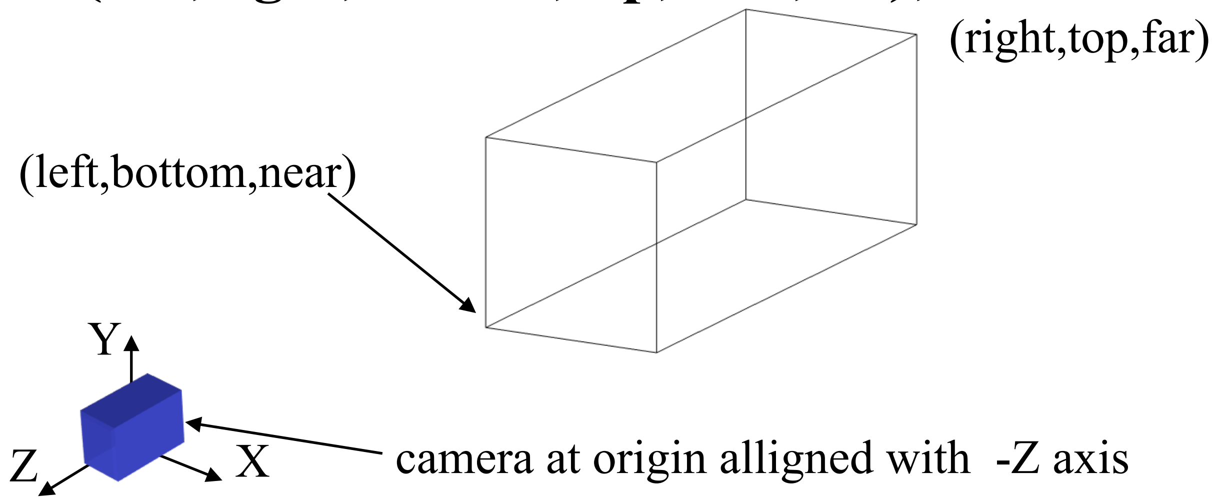
- position/orientation
- projection
- field-of-view

OpenGL supports two projection models: orthographic & perspective

Orthographic projection: All rays parallel

- Default camera is at the origin alligned with the -Z axis
- Projection is specified by a parallelapiped as:

glOrtho(left,right,bottom,top,near,far);



Utility Libraries

A number of related libraries are available which provide utility functions:

Graphics Utility Library (GLU): All OpenGL implementations

Common objects (sphere,cylinder..)

Uses only GL library common

All commands begin **'glu'**

#include "GL/glu.h"

GL Utility Toolkit (GLUT): Separate from OpenGL

Common interface with windows systems

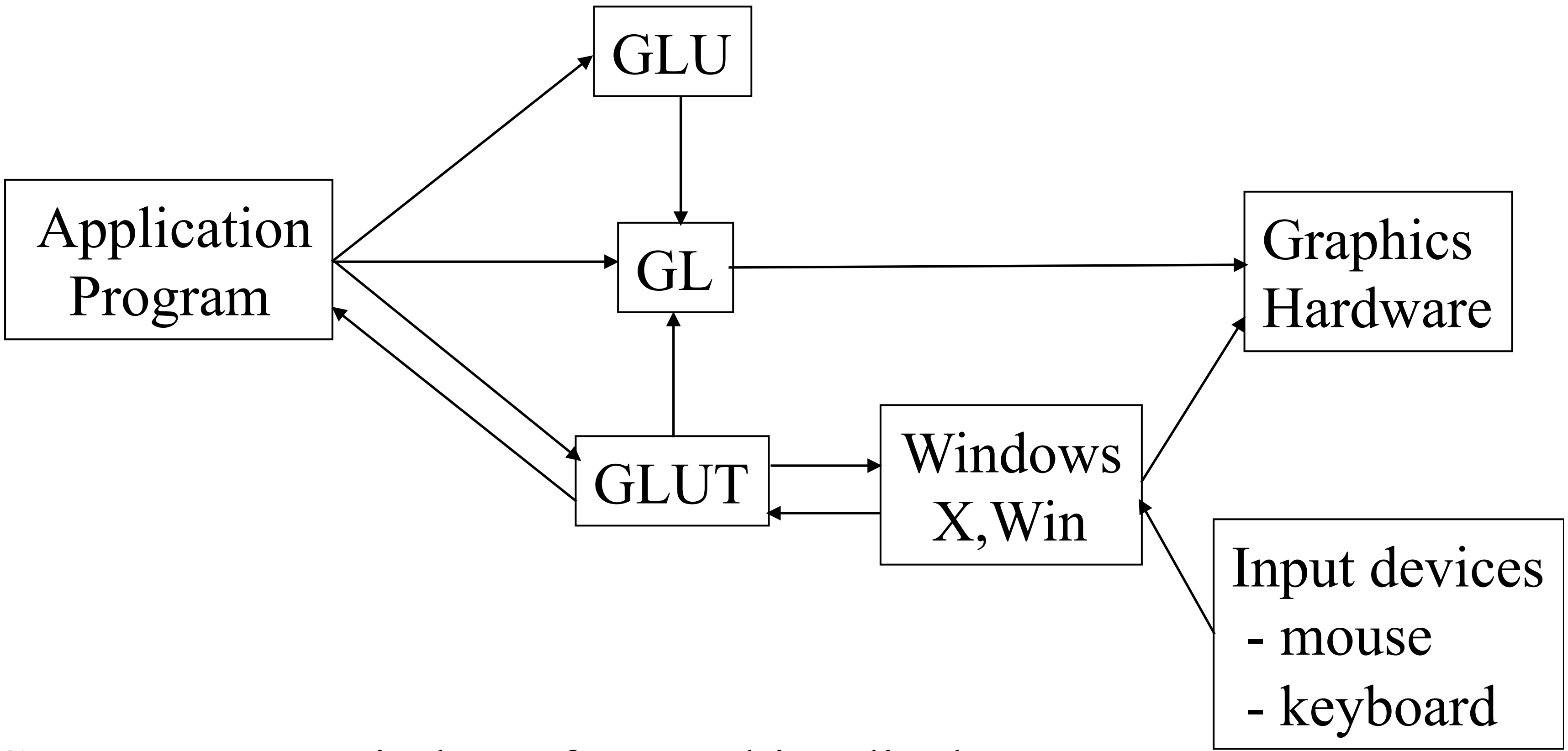
Versions for (Xwindows, Microsoft Windows….)

Minimum functionally for a windows system

All commands begin **'glut'**

#include "GL/glut.h"

Library Organisation



GLUT - setup windows for graphics display
- input events from mouse/keyboard

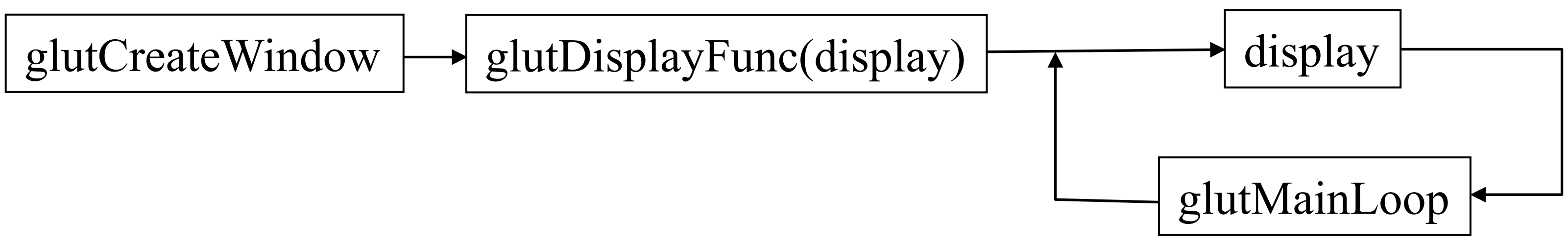
Getting Started with GLUT

GLUT provides useful utility functions for implementing a graphics application:

glutCreateWindow() - creates a window of a pre-specified size.

glutDisplayFunc(display) - calls a user specified function "display" whenever window needs to be drawn

glutMainLoop() - enter an event processing loop so that graphics application continues to run & respond to user input until exited



GLUT main function

```
#include “GL/glut.h”        /* include GLUT,GLU,GL */

int main(int argc, char **argv){
       glutInit(&argc,argv);        /* initialise glut */

       /* initialise OpenGL display state */
       glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

       /* initialise window */
       glutInitWindowSize(500,500);
       glutInitWindowPosition(0,0);
       glutCreateWindow(“simple OpenGL example”);

       /* register display function */
       glutDisplayFunc(display);

       init();  /* call my own initialisation routine */

       /* start displaying & event handling*/
       glutMainLoop();
       return 0;
}
```

Example: Completing the Square

We can now write the display functions for drawing a square

```
void init() {
      glClearColor(1.0,0.0,0.0,0.0); /* background color */
      glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0); /* viewing */
}

void display() {
      glClear(GL_COLOR_BUFFER_BIT); /* clear window */
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_POLYGON);
            glVertex3f(0.25,0.25,0.0);
            glVertex3f(0.75,0.25,0.0);
            glVertex3f(0.75,0.75,0.0);
            glVertex3f(0.25,0.75,0.0);
      glEnd();
      glFlush();    /* draw everything */
}
```

Example 2: The Sierpinski Gasket

Sierpinski gasket is a fractal shape defined by a simple recursive algorithm:

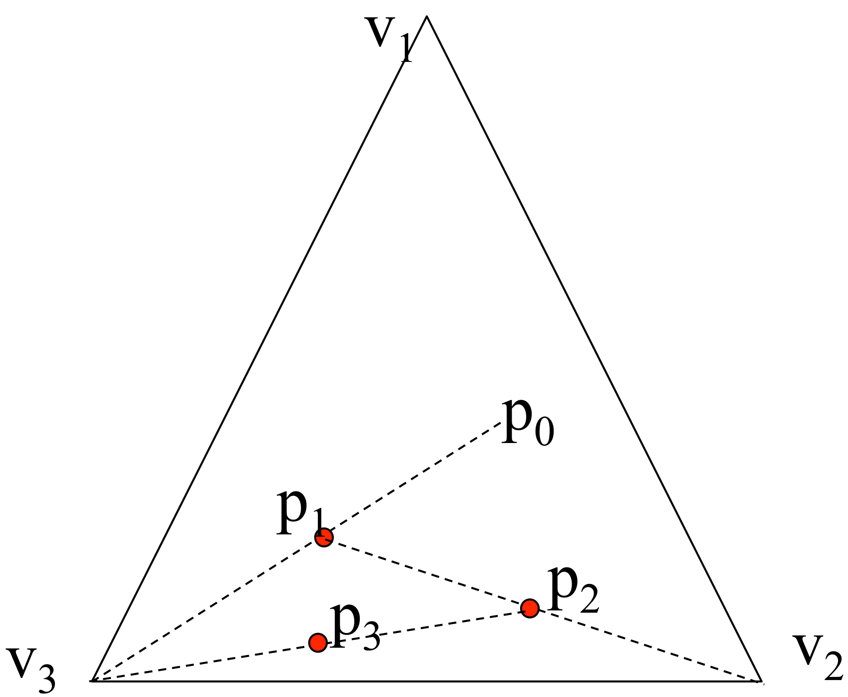
(1) pick 3 triangle verticies v_1, v_2, v_3

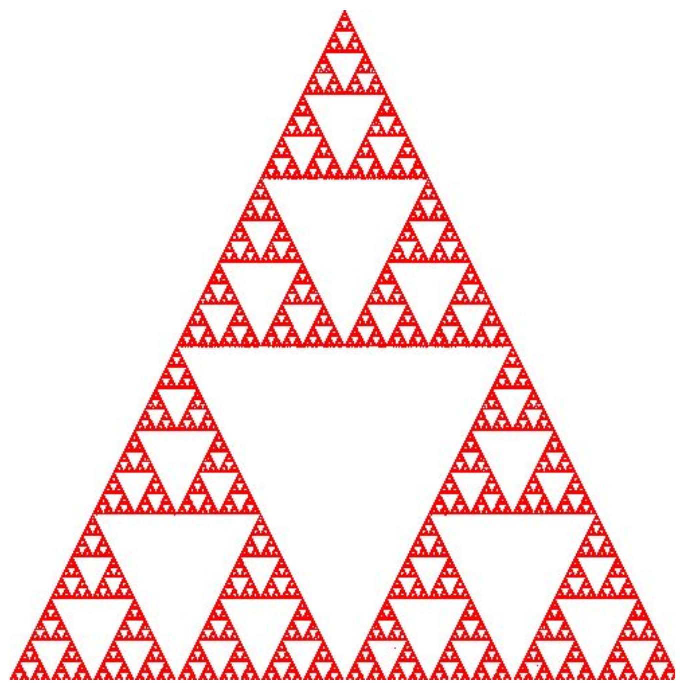
(2) select a point inside the triangle p

(3) randomly pick a triangle vertex v_i

(4) draw the point p' halfway between p and v_i

(5) repeat 3 & 4 with p = p'





Implementation of Sierpinski Gasket

```
void display(void) {
        typedef GLfloat point2[2];                          /* define2d  point type */
        point2 verticies[3]={{0.0,0.0},{250.0,500.0}, {500.0,0.0}};  /* a triangle */
        point2 p = {75.0,50.0};                             /* arbitrary start point */
        int j,k;

        glClear(GL_COLOR_BUFFER_BIT); /* clear window */
        /* Sierpinski algorithm: Recursive plotting of 5000 points */
        for (k=0; k<5000; k++) {
                j=rand()%3;                                 /* pick vertex at random */
                p[0] = (p[0]+verticies[j][0])/2.0;   /* new half-way point */
                p[1] = (p[1]+verticies[j][1])/2.0;
                glBegin(GL_POINTS);          /* add point to display list */
                    glVertex2fv(p);
                glEnd();
        }
        glFlush();  /* display now */
}
```