Interaction Handling

- **Input-output devices**

  - Portability: graphic packages should be device independent

  - Graphics output

    * Primitives are specified in terms of an abstract integer coordinate system

    * The application does not need to set individual pixels in the frame buffer

  - Graphics input

    * It supports a set of logical input devices

    * The application does not need to know the details of the physical input devices

    * Locator: for specifying the screen coordinates and the state of one or more associated buttons

    * Keyboard: for specifying character string input

    * For each input device, an event trigger (e.g, a button push) and a measure (e.g., location of the mouse) is defined

- **Modes of interacting with logical devices**

  - Sampling

    * The application queries the current value of a logical input device and continues execution

    * Sampling takes place regardless of any user activity

    * Costly for interactive applications (CPU time is wasted)

  - Interrupt-driven

    * Idea: respond only when the user acts

    * The application enables one or more input devices and continues

execution until interrupted by some input event.

* When an input occurs, control passes asynchronously to an interrupt procedure which responds to the event.

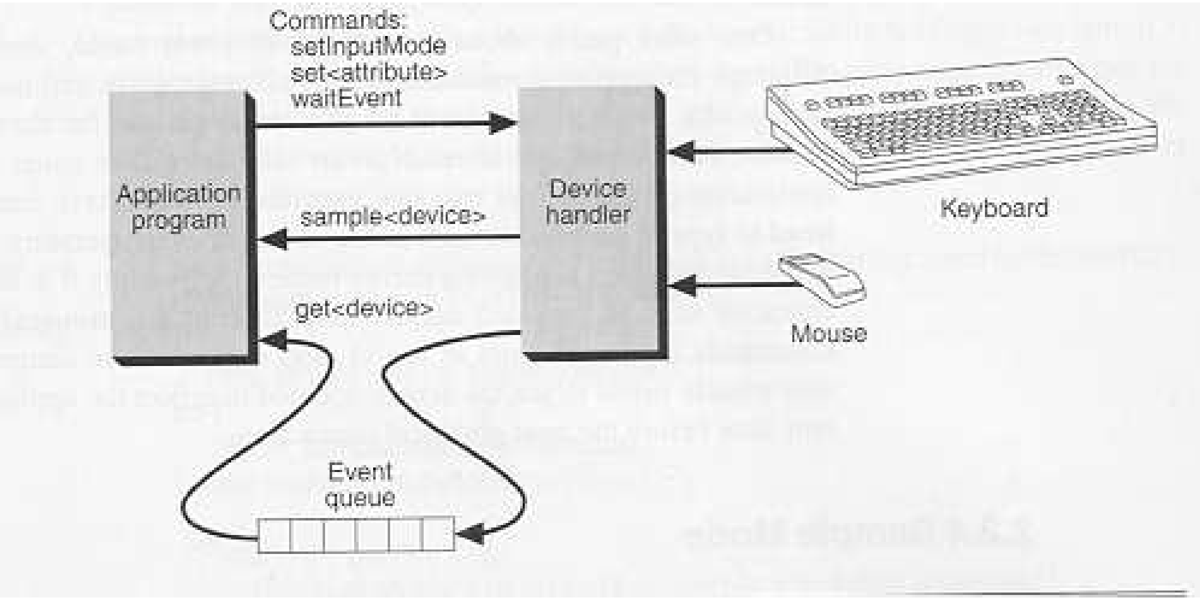
- Event Driven

* A synchronous simulation of interrupt-driven.

* The application enables the devices and continues execution

* The devices are monitored in the background, by storing information about each event in an event FIFO queue

* If the queue is empty, the application can either go to a wait state or continue execution



Generate the initial image;
Activate interactive device(s) in event mode;
do { / main event loop */*
  wait for user-triggered event on any of several devices;
  switch (which device caused event) {
    case DEVICE_1: collect event measure data, process, respond;
    case DEVICE_2: collect event measure data, process, respond;
    ...
  }
} while (user does not request quit);

Interaction handling using GLUT

- **Entering an event-processing loop**

void glutMainLoop(void)

- **Callback functions**

- A function is associated with each type of event

* Write a function which performs the desired actions when a specific event occurs

* Register the function by passing its name as an argument to a function of the form:

glut<name>Func()

- Mouse events

* Callback function: *void mouse_callback(int button, int state, int x, int y)*

* Registering function (buttons): *glutMouseFunc(mouse_callback)*

* Registering function (draging): *glutMotionFunc(mouse_callback)*

- Keyboard events

* Callback function: *void keyboard_callback(usigned char key, int x, int y)*

* Registering function: *glutKeyboardFunc(keyboard_callback)*

- Window events (resizing)

* Callback function: *void reshape_callback(int width, int height)*

* Registering function: *glutReshapeFunc(reshape_callback)*

- Display callback

* Callback function: *void display_callback(void)*

* Registering function: *glutDisplayFunc(display_callback)*