S J P N Trust's

Hirasugar Institute of Technology, Nidasoshi. *Inculcating Values, Promoting Prosperity*

Approved by AICTE, Recognized by Govt. of Karnataka and Affiliated to VTU Belagavi.

Accredited at 'A' Grade by NAAC

Programmes Accredited by NBA: CSE, ECE, EEE & ME

Module-5

18CS62

Input& interaction, Curves and Computer Animation:

Course Outcome

Explain curve generating concepts and interactive computer graphics using the OpenGL

Prof. Rahul Palakar

Input and Interaction

Objectives are to learn about:

- Introducing variety of devices that are used for interaction.
- Learn about two different perspectives from which the input devices are considered:

1) The way that the physical devices can be described by the real-world properties,

2) The way that these devices appear to the application program.

Input Devices

There are two different ways to look at the devices:

- Physical devices: keyboard or mouse, and discuss how they work
- Logical devices: the way application programs look at the devices.

A logical device is characterized by its high-level interface with the user program, rather than by its physical characteristics.

Logical Devices

The main characteristics that describes the logical behavior of an input device:

1) what measurements the device returns to the user program, and

2) when the device returns those measurements.

In general, there are six classes of logical input devices:

1. String – provides ASCII strings to the user program (logical implemented via keyboard)

2. Locator – provides a position in world coordinates to the user program (pointing devices and conversions may be needed)

Logical Devices

3. Pick – returns the identifier of an object to the user program. (pointing devices and conversions may be needed)

4. Choice – allows users to select one of the distinct number of options (widgets – menus, scrollbars, and graphical buttons)

5. Dial – provides analog input to the user program (widgets –slidebars,…)

6. Stroke – it returns an array of locations (similar to multiple use of a locator, continuous)

Measure and Trigger

- The manner by which physical and logical input devices provide input to an application program can be described in terms of two entities:

1) A measure process, and 2) A device trigger.

❖The **measure** of a device is what the device returns to the user program.

❖The **trigger** of a device is a physical input on the device with which the user can signal the computer.

Measure and Trigger

Example1:

- The measure of a keyboard is a string,
- The trigger could be the “return” or “enter” key.

Example2:

- For a locator the measure includes the location and
- The trigger can be the button on the pointing device.

Input Modes

In addition to multiple types of logical input devices, we can obtain the measure of a device in three distinct modes:

1) Request mode,

2) Sample mode, and

3) Event mode.

- It defined by the relationship between the measure process and the trigger.
- Normally, the initialization of an input device starts a measure process.

Input Modes

1) Request mode: In this mode the measure of the device is not returned to the program until the device is triggered.

- A locator can be moved to different point of the screen. The Windows system continuously follows the location of the pointer, but until the button is depressed, the location will not be returned.

Input Modes

2) Sample mode: Input is immediate. As soon as the function call in the user program is encountered, the measure is returned, hence no trigger is needed.

- For both of the above modes, the user must identify which devices is to provide the input.

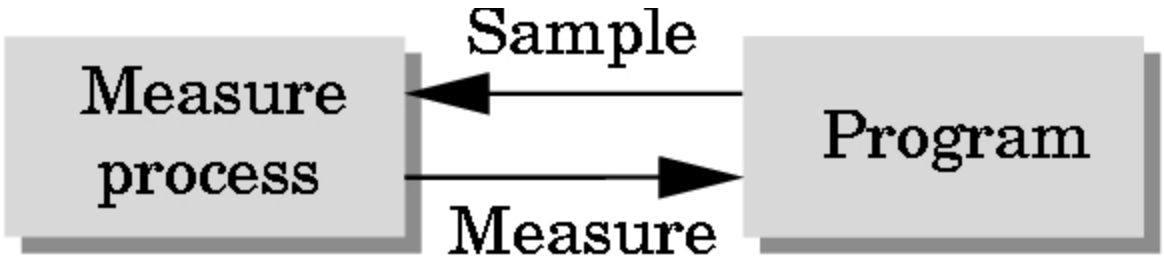
request_locator(device_id, &measure);

Input Modes

sample_locator(device_id, &measure);

identifier location

- *Think of a flight simulator with many input evices*



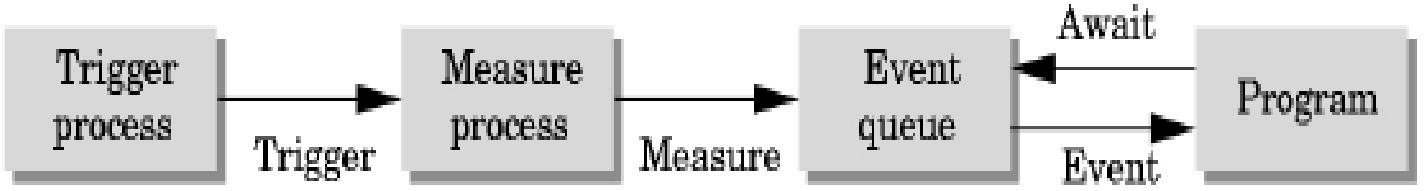
Input Modes

3) Event mode: The previous two modes are not sufficient for handling the variety of possible human-computer interactions that arise in a modern computing environment. They can be done in three steps:

1) Show how event mode can be described as another mode within the measure trigger paradigm.
2) Learn the basics of client-servers when event mode is preferred, and
3) Learn how OpenGL uses GLUT to do this.

Input Modes

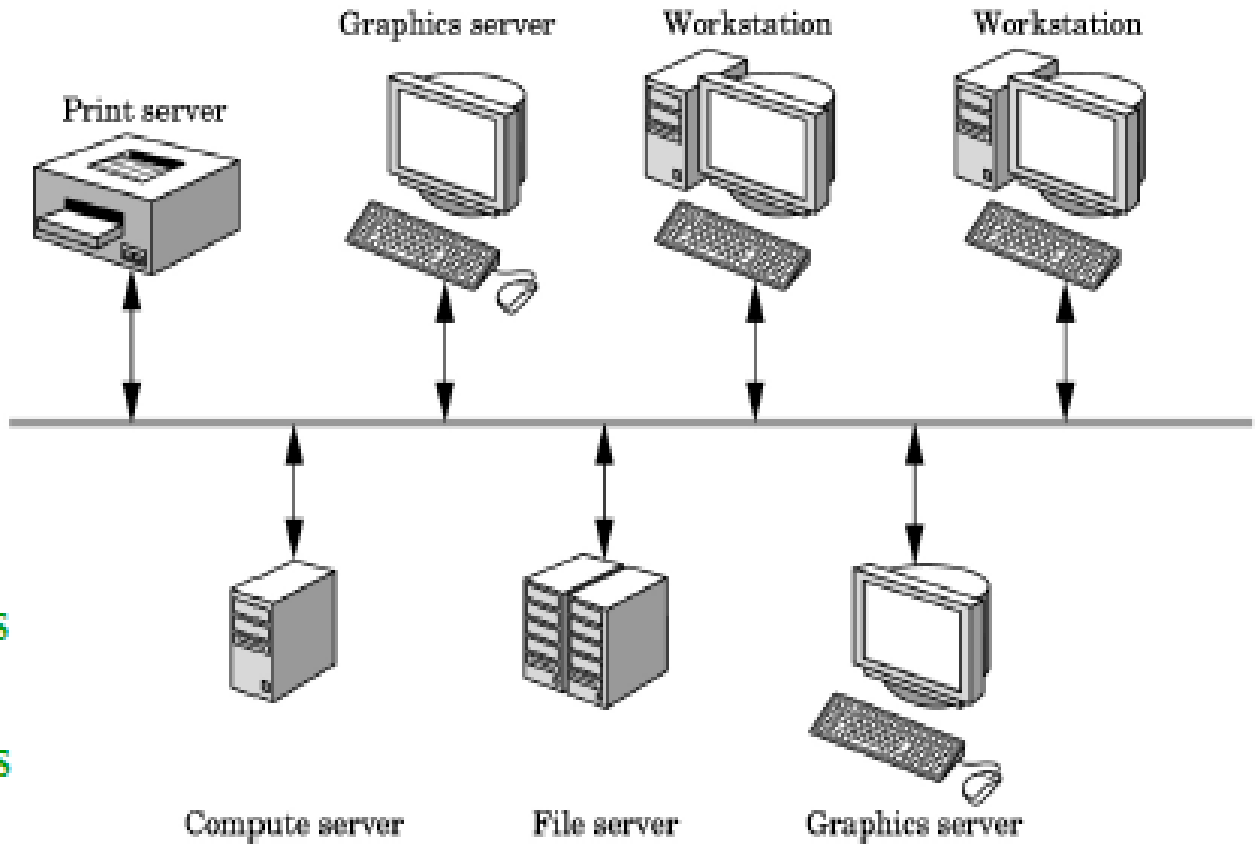
- In an environment with multiple input devices, each with its own trigger and each running a measure process.
- Each time that a device is triggered, an **event** is generated. The device measure, with the identifier for the device, is placed in an **event queue.** The user program executes the events from the queue.
- When the queue is empty, it will wait until an event appears there to execute it.
- Another approach is to associate a function called a **callback** with a specific type of event. This is the approach we are taking.



Client and Servers

If a computer graphics is to be useful for a variety of real applications, it must function well in a world of distributed computing and networks.

In this world, the building blocks are entities called **servers** that can perform tasks for **clients**.



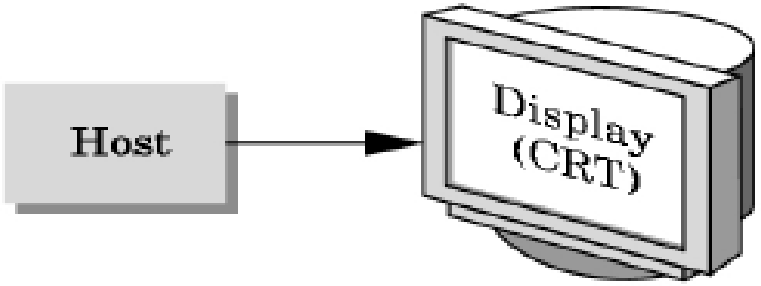
- Print servers
- Compute servers
- File servers
- Terminal servers

Display Lists

Display Lists

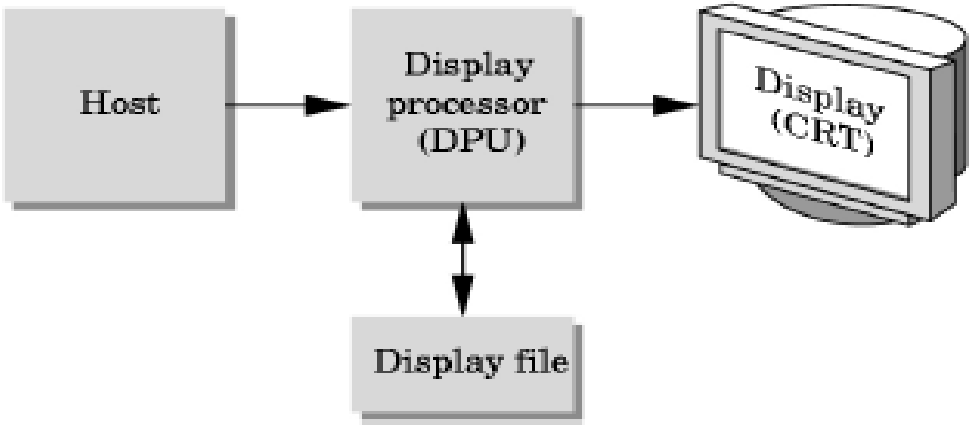
Display lists illustrate how we can use clients and servers on a network to improve graphics performance.

The computer would send out the necessary information to redraw the display at a rate sufficient to avoid noticeable flicker. In the past, since computers were slow and expensive, the cost of keeping even a simple display refreshed was prohibitive for all but a few applications.



The solution is to this problem was to build a special-purpose computer, called a **display processor**.

Today, the display processor is a a graphics server, and the user program on the host computer is a client.



Display List

We can send graphical entities to a display in one of the two ways:

1) Send the complete description of our objects to the graphics server.

For a typical geometric primitives, this transfer consists of; sending vertices, attributes, and primitive types, in addition to viewing information.

2) Define the object once, then put its description in a display list. The display list is stored in the server and redisplayed by a simple function call issued from the client to the server. This method is called **retained mode** graphics.

Disadvantages associated with the use of display list:

1) Display lists require memory on the server,

2) There is an overhead for creating a display list.

Definition and Execution of Display Lists:

- Each time we wish to draw the box on the server, we execute the function: *glCallList(BOX);*

Note that the present state of the system determines which transformations are applied to the primitives in the display list. Thus, if we change the model-view or projection matrices between executions of the display list, the box will appear in different places or even will no longer appear.

Create one display list

```
GLuint index = glGenLists(1);
// compile the display list, store a triangle in it
glNewList(index, GL_COMPILE);
        glBegin(GL_TRIANGLES);
                glVertex3fv(v0);
                glVertex3fv(v1);
                glVertex3fv(v2);
        glEnd();
glEndList();
... // draw the display list
glCallList(index);
... // delete it if it is not used any more
glDeleteLists(index, 1);
```

Example:

```
glMatrixMode(GL_PROJECTION)
for(i = 1; i < 5; i++)
   {
      glLoadIdentity( );
      gluOrtho2D( -2.0*i , 2.0*i , -2.0*i , 2.0*i );
     glCallList(BOX);
   }
```

- Every time that the *glCallList* is executed, the box is redrawn with a different clipping rectangle.

Programming Event-Driven Input

Using the Pointing Device:

Two types of events are associated with the pointing device.

- **move event:** is generated when the mouse is moved with one of the buttons depressed, for a mouse the **mouse event** happens when one of the buttons is depressed or released.
- **passive move event:** is generated when the mouse is moved without a button being hold down.

The mouse callback function looks like this:

glutMouseFunc(mouse_callback_func)

void mouse_callback_func(int button, int state, int x, int y)

Within the callback function, we define what action we want to take place if the specified event occurs. There may be multiple actions defined in the mouse callback function corresponding to the many possible button and state combinations.

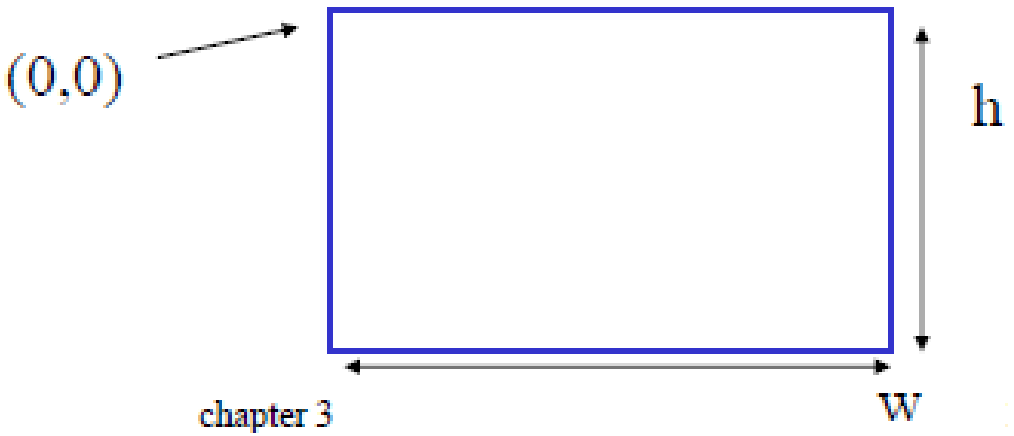
Using the Pointing Device

- Suppose we want the program to terminate when the left button is depressed.

```
void mouse_callback_function(int button, int state, int x, int y)
    {
        if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        exit( 1);
    }
```

Positioning

- The position in the screen window is usually measured in pixels with the origin at the top-left corner
  - Consequence of refresh done from top to bottom
- OpenGL uses a world coordinate system with origin at the bottom left
  - Must invert y coordinate returned by callback by height of window
  - $y = h - y;$



Keyboard Events

- We can use the keyboard event as an input device. Keyboard events are generated when the mouse is in the window and one of the keys is pressed.
- In GLUT, there callback for the release of a key.

glutKeyboardFunc(keyboard);

glutKeyboardUpFunc(keyboard);

To use the keyboard to exit a program:

```
void keyboard(unsigned char key, int x, int y)
        {
                if(key == 'q' || key == 'Q')
                exit(1);
        }
```

Window Management

GLUT supports both multiple windows and subwindows of a given window.

id = glutCreateWindow(“Second Window”);

- The returned integer value allows us to select this window as the current window:

glutSetWindow(id);

Menus

We can use our graphics primitives and our mouse callback to construct various graphical input devices.

GLUT provides **pop-up menus**.

- Using menus involves:

1. Must define the entries in the menu,
2. must link the menu to a particular mouse button, and
3. must define a callback function corresponding to each menu entry.

Example – Pop-up menu

```
glutCreateMenu(demo_menu);
glutAddMenuEntry("quit", 1);
glutAddMenuEntry("increase square size", 2);
glutAddMenuEntry("decrease square size", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

The callback function looks like this:

```
void demo_menu(int id)
        {
                if(id == 1) exit(1);
                else if (id == 2) size = 2*size;
                else size = size / 2;
                glutPostRedisplay( );
        }
```

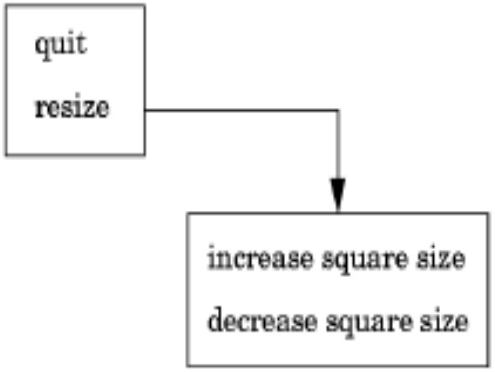
Hierarchical menu

Suppose we want the main menu that we create to have two entries:

1) the first one to terminate the program

2) the second to pop-up a submenu.

```
Sub_menu = glutCreateMenu(size_menu);
glutAddMenuEntry("increase  square size", 2);
glutAddMenuEntry("decrease  square size", 3);
glutCreateMenu(top_menu);
glutAddMenuEntry("quit", 1);
glutAddSubMenu("Resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```



Now we have to write the call back functions, *size_menu* and *top_menu*.

Picking

- Picking is an input operation that allows the user to identify an object on the display.
- Picking is done by a pointing device, the information returned to the application program is not a position.

A pick device is more difficult to implement than the locator device. There are two ways to do this:

1) **selection**, involves adjusting the clipping region and viewport such that we can keep track of which primitives in a small clipping region are rendered into a region near the cursor. Creates a **hit list**.

2) **bounding rectangles** or **extents,** this is the smallest rectangle, aligned with the coordinates axes, that contains the object.

Rendering Modes:

OpenGL can render in one of three modes selected by **glRenderMode(mode)**

1. GL_RENDER: normal rendering to the frame buffer (default)
2. GL_FEEDBACK: provides list of primitives rendered but no output to the frame buffer
3. GL_SELECTION: Each primitive in the view volume generates a hit record that is placed in a name stack which can be examined later

Selection Mode Functions:

- glSelectBuffer(GLsizei n, GLuint *buff): specifies name buffer
- glInitNames(): initializes name buffer
- glPushName(GLuint name): push id on name buffer
- glPopName(): pop top of name buffer
- glLoadName(GLuint name): replace top name on buffer

id is set by application program to identify objects

Using Selection Mode:

1. Initialize name buffer
2. Enter selection mode (using mouse)
3. Render scene with user-defined identifiers
4. Reenter normal render mode
    - This operation returns number of hits
5. Examine contents of name buffer (hit records)
    - Hit records include id and depth information

```
void mouse (int button, int state, int x, int y)
        {
                GLUint nameBuffer[SIZE];
                 GLint hits;
                GLint viewport[4];
                if (button == GLUT_LEFT_BUTTON
                && state== GLUT_DOWN)
                { /* initialize the name stack */
                glInitNames();
                glPushName(0);
                glSelectBuffer(SIZE, nameBuffer)
```

```
/* set up viewing for selection mode */
glGetIntegerv(GL_VIEWPORT, viewport);
//gets the current viewport
glMatrixMode(GL_PROJECTION);
 /* save original viewing matrix */
glPushMatrix();
glLoadIdentity();
/* N X N pick area around cursor */
gluPickMatrix( (GLdouble) x, (GLdouble)(viewport[3]-
                              y), N,N,viewport);
```

```
/* same clipping window as in reshape callback */
        gluOrtho2D(xmin,xmax,ymin,ymax);
        draw_objects(GL_SELECT);
        glMatrixMode(GL_PROJECTION);
        /* restore viewing matrix */
        glPopMatrix();
        glFlush();
        /* return back to normal render mode */
        hits = glRenderMode(GL_RENDER);
```

```
/* process hits from selection mode rendering*/
        processHits(hits, nameBuff);
        /* normal render */
        glutPostRedisplay();
         }
}
void draw_objects(GLenum mode)
{
        if (mode == GL_SELECT)
        glLoadName(1);
        glColor3f(1.0,0.0,0.0)
        glRectf(-0.5,-0.5,1.0,1.0);
```

```
        if (mode == GL_SELECT)
        glLoadName(2);
        glColor3f(0.0,0.0,1.0)
        glRectf(-1.0,-1.0,0.5,0.5);
}
void processHits(GLint hits, GLUint buffer[])
{
        unsigned int i,j;
}
```

Bezier Curve

- So a Bezier curve is a mathematically defined curve used in two-dimensional graphic applications like abode illustrator, inkscape etc. The curve is defined by four points: **the initial position** and **the terminating position** i.e **P0** and **P3** respectively (which are called “anchors”) and **two separate middle points** i.e **P1** and **P2**(which are called “handles”) in our example. Bezier curves are frequently used in computer graphics, animation, modeling etc.

How do we Represent Bezier Curves Mathematically

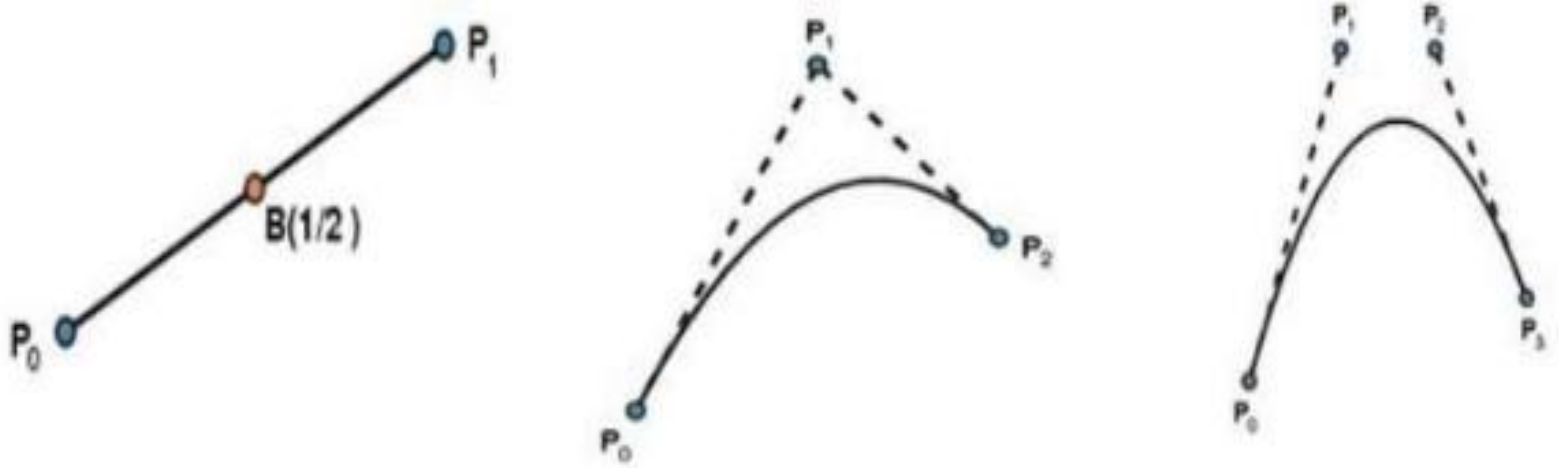
Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as –

$$\mathrm{P(u)} = \sum_{i=0}^{n} P_i B_i^n(u)$$

Where P_i is the set of points and $B_i^n(u)$ represents the Bernstein polynomials i.e. Blending Function which are given by –

$$B_i^n(u) = \binom{n}{i}(1-u)^{n-i}u^i$$

- Where **n** is the polynomial order, **i** is the index, and **u/t** is the variable which have values from **0** to **1**.



- For cubic bezier curve order(n) of polynomial is **3** , index(i) vary from **i = 0 to i = n i.e. 3** and u will vary from 0<=u<=1 .

$$P(u) = P_0 B_0^3(u) + P_1 B_1^3(u) + P_2 B_2^3(u) + P_3 B_3^3(u)$$

Cubic Bezier Curve blending function are defined as :

$$B_0^3(u) = \binom{3}{0}(1-u)^{3-0}u^0 \equiv 1(1-u)^3u^0$$

$$B_1^3(u) = \binom{3}{1}(1-u)^{3-1}u^1 \equiv 3(1-u)^2u^1$$

$$B_2^3(u) = \binom{3}{2}(1-u)^{3-2}u^2 \equiv 3(1-u)^1u^2$$

$$B_3^3(u) = \binom{3}{3}(1-u)^{3-3}u^3 \equiv 1(1-u)^0u^3$$

So

$$P(u) = (1-u)^3 P_0 + 3u^1(1-u)^2 P_1 + 3(1-u)^1 u^2 P_2 + u^3 P_3$$

and

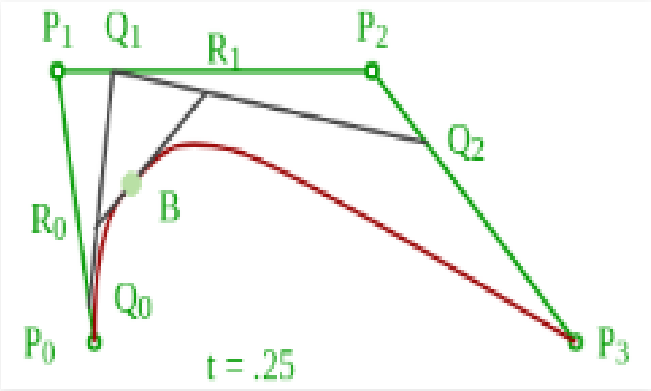
$$P(u) = \{x(u), y(u)\}$$

Now,

$$x(u) = (1-u)^3 x_0 + 3u^1(1-u)^2 x_1 + 3(1-u)^1 u^2 x_2 + u^3 x_3$$

$$y(u) = (1-u)^3 y_0 + 3u^1(1-u)^2 y_1 + 3(1-u)^1 u^2 y_2 + u^3 y_3$$

So we will calculate curve x and y pixel by incrementing value of u by **0.0001**.



Bezier Curve Properties

- The first and last control points are interpolated.
- The tangent to the curve at the first control point is along the line joining the first and second control points.
- The tangent at the last control point is along the line joining the second last and last control points.
- The curve lies entirely within the convex hull of its control points.
- They can be rendered in many ways.

- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.