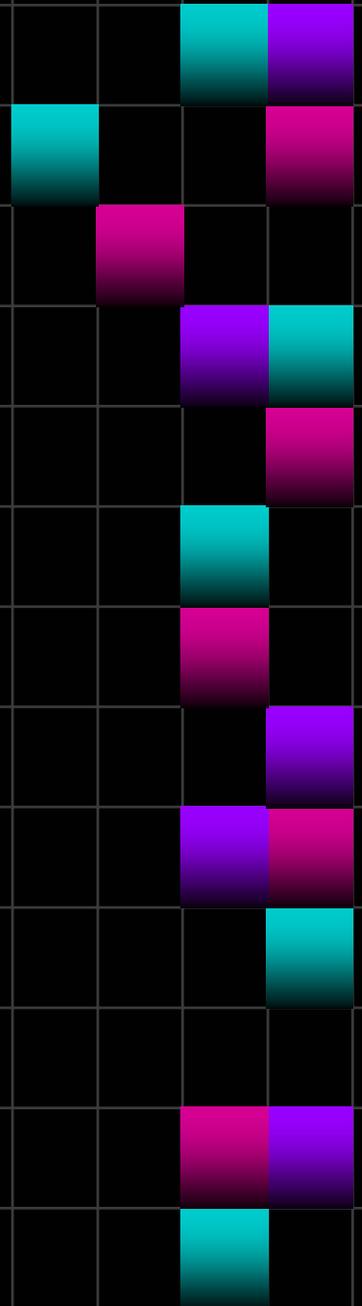


Graphics Animation Using Borland's bgi

Mr. Dave Clausen
La Cañada High School



Graphics Commands for the bgi

- Remember, a summary of Borland's graphics commands for Borland C++ 3.0 for DOS and Borland C++ for Windows can be found in a previous lecture.

Animation basic concepts.

- The basic concept for graphics animation is to have some form of loop (definite or indefinite), where we **draw an image, erase it** (by drawing it exactly again, only in the background color instead of the foreground color), and **update its coordinates**.
- This process repeats until the image reaches a desired location, or until some other condition is met.

Blinking Objects

- The algorithm for a blinking object is as follows:

For a definite number of iterations do (or while)

 Draw the object (in a foreground color)

 Pause for a specified time (as necessary)

 Erase the object (draw in the background color)

- If you wish the image to remain at the end, draw it again after the loop.

[blink.cpp](#)

[blink.txt](#)

[blink.exe](#)

Horizontal motion

- The algorithm for horizontal motion is:
 - While the image is not at its destination (or for)
 - Draw the image in the foreground color(s)
 - Pause for a specified time (use symbolic constants)
 - Erase the image (draw it in the background color)
 - Update the image's position
 - To update the image's position, increment or decrement:
 - $x = x + \text{delta_x}$; or $x = x - \text{delta_x}$;
 - If you are using a for loop, this is taken care of already.
- If you wish the image to remain at the end, draw it again after the loop.

Vertical motion

- The algorithm for vertical motion is:
 - While the image is not at its destination (or for)
 - Draw the image in the foreground color(s)
 - Pause for a specified time (use symbolic constants)
 - Erase the image (draw it in the background color)
 - Update the image's position
 - To update the image's position, increment or decrement:
 - $y = y + \text{delta_y}$; or $y = y - \text{delta_y}$;
 - If you are using a for loop, this is taken care of already.
- If you wish the image to remain at the end, draw it again after the loop.

Linear motion

- The algorithm for linear motion is:
 - While the image is not at its destination (or for)
 - Draw the image in the foreground color(s)
 - Pause for a specified time (use symbolic constants)
 - Erase the image (draw it in the background color)
 - Update the image's position
 - To update the image's position, increment or decrement:
 - $y = y + \text{delta_y}$; or $y = y - \text{delta_y}$; or
 - $x = x + \text{delta_x}$; or $x = x - \text{delta_x}$;
- If you wish the image to remain at the end, draw it again after the loop.

Linear motion: y in terms of x

- The algorithm remains the same as defined on previous slides. The difference is how we calculate the variable delta_y in terms of x .
- For linear paths we may use the increment:

$$y = y + (3 * x - 5);$$

//The equation of a line between 2 points is:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1)$$

[mvsquare.cpp](#)

[mvsquare.txt](#)

[mvsquare.exe](#)

Non linear motion

- The algorithm remains the same as defined on previous slides. The difference is how we calculate the variable `delta_y` in terms of `x`.
- For parabolic paths we may use the function:

```
y = y + (0.01 * (x * x));
```

```
//The basic formula for a parabola in standard form:
```

$$y = a(x - h)^2 + k$$

[parabola.cpp](#)

[parabola.txt](#)

[parabola.exe](#)

To Move an Object in a Parabolic Path:

- Let's look at this example which doesn't merely draw a parabola, but moves an object in a parabolic path.

[paracirc.cpp](#)

[paracirc.txt](#)

[paracirc.exe](#)

Other Examples:

- Let's look at examples of previous student's work.
- They fall into 2 categories:
 - non-interactive animation
 - as previously discussed
 - interactive animation
 - using indefinite loops while not `kbhit()`.
 - Keys can be detected using `getch()`
 - Using `switch` or nested ifs to determine action of key hit. (use ASCII codes of the keys.)