Structured Programming
*Using C++*

Lecture 10: Graphics Programming with the Dark GDK Library

*Dr. Amal Khalifa*

---

**Lecture Contents:**

- the Dark GDK Library
- The Screen Coordinate System
- Basic Graphics Shapes
- Colors
- Game loop and animation
Concept:

All C++ programs that use the Dark GDK library start out with the same code. The first step in learning to write a graphics program is to learn how to start a Dark GDK program in C++.

Program skeleton

Minimum framework of a C++/Dark GDK program

```cpp
#include "DarkGDK.H"

void DarkGDK()
{
    // Statements written here are executed when the program runs.
}
```

---

Dr. Amal Khalifa, 2012
Example

Program 2-2  (EmptyWindow2.cpp)

```cpp
1   // This program displays an empty window.
2   #include "DarkGDK.h"
3
4   void DarkGDK ()
5   {
6       // Pause the program until the user presses a key.
7       dbWaitKey();
8   }
```

- Dark GDK function names start with the letters `db`
- The Game Creators, the software company that created the Dark GDK library, have also created a programming language called `Dark BASIC`
  - Dark GDK function names start with the letters `db`, meaning `Dark BASIC`

The Screen Coordinate System

**Concept:**
A system of X and Y coordinates is used to identify the locations of pixels in a window.

- The images that are displayed on a computer screen are made up of tiny dots called `pixels`
- The default Dark GDK window is 640 pixels wide and 480 pixels high
- It has a `resolution` of 640 by 480
The Screen Coordinate System

- Coordinates are written in the form \((X, Y)\)

- Example:
  - Upper-left corner pixel coordinates are \((0, 0)\)
  - Lower-right corner pixel coordinates are \((639, 479)\)
  - Any points that have coordinates outside these ranges are not visible in the window

Each pixel has

- **X coordinate**: Identifies the horizontal position (increase from left to right)
- **Y coordinate**: Identifies the vertical position (increase from top to bottom)
### Changing the Size of the Program Window

- Use the `dbSetDisplayMode` function:
  
  ```c
  dbSetDisplayMode(150, 100, 32);
  ```

- Changes size of program's window
  - **Width**
    - The width, in pixels, of the program's window
  - **Height**
    - The height, in pixels, of the program's window
  - **Color depth**
    - Integer specifying the number of bits used to store the color displayed in the window

### Changing Window Title

- The `dbSetWindowTitle` function displays text in the window's title bar
  
  ```c
  dbSetWindowTitle(string);
  ```
Drawing Dots

- The `dbDot` function draws a dot at a specific pixel location in the Dark GDK window.

\[ \text{dbDot}(x, y); \]

- The `x` argument is the X coordinate
- The `y` argument is the Y coordinate
- Example:
  - draw a dot at the X coordinate 319 and the Y coordinate 239

\[ \text{dbDot}(319, 239); \]

The `dbWait` Function

- The `dbWait` function causes the program to wait for a specified amount of time before continuing
- Here is the general format of how you call the `dbWait` function:

\[ \text{dbWait}(\text{time}); \]

- The value that you provide for the `time` argument is the number of milliseconds you want the program to wait.
  - There are 1000 milliseconds in a second
Basic 2D Shapes

- The Dark GDK library contains several functions for drawing basic 2D shapes:
  - Lines
  - Circles
  - Ellipses
  - Rectangles

- Objects that appear in 2D have only two dimensions:
  - Width
  - Height

Drawing Lines

```cpp
dbLine(x1, y1, x2, y2);
```

- Example:
  ```cpp
dbLine(80, 120, 400, 520);
  ```
Drawing Rectangles

- The `dbBox` function draws a filled rectangle
  - Filled means it is filled with color
  
  \[
  \text{dbBox}(x_1, y_1, x_2, y_2);
  \]
  - \(x_1, y_1\): coordinates for the rectangle's upper-left corner
  - \(x_2, y_2\): coordinates for the rectangle's lower-right corner

\[
\text{dbBox}(100, 80, 540, 380)
\]

Drawing Circles

- The `dbCircle` function draws a circle
  - The radius argument specifies the circle's radius
  - The radius is the distance, in pixels, from the center point to the outer edge

\[
\text{dbCircle}(x, y, \text{radius});
\]

\[
\text{dbCircle}(320, 240, 100)
\]
### Drawing Ellipses

- The `dbEllipse` function draws an **ellipse**
  - an ellipse is an oval shape
  - `dbEllipse(x, y, xrad, yrad);`
    - `x` and `y`: the coordinates of the ellipse’s center point
    - `xrad`: the radius along the X axis
    - `yrad`: the radius along the Y axis

```
dbEllipse(320, 240, 140, 100);
```

### Drawing Tips!!

- Points that are above the top row of pixels in the window have a negative Y coordinate
- Points that are to the left of the leftmost column of pixels have a negative X coordinate

```
dbCircle(-50, -20, 200);
```
Displaying Text

Concept:

You can use the `dbPrint`, `dbText`, or `dbCenterText` functions to display text in the Dark GDK window. You can use the `dbSetWindowTitle` function to display text in the window’s title bar.

The `dbPrint` function displays a `string` of characters

```
dbPrint(string);
```

- If no argument is passed
  - a blank line will be displayed
- The first call to the `dbPrint` function
  - output is printed at the top of the window, justified along the left side
- Each subsequent call to the `dbPrint` function
  - prints a line of output below the previous line of output.
Displaying Text

- The `dbText` function displays a string of characters at a specific location in the window
  
  $\text{dbText}(x, y, \text{string});$

- When the string is displayed
  
  - The upper-left corner of the first character will be positioned at the $X$ and $Y$ coordinates

  $\text{dbText}(10, 10, \text{“Hello World”});$

Displaying Text

- Text can be centered horizontally with the `dbCenterText` Function

  $\text{dbCenterText}(319, 239, \text{“Game Over”});$
The Program Development Cycle

Concept:
When creating programs, programmers typically follow a process known as the program development cycle.

Example
1. The user entered 450 and pressed Enter.
2. The user entered 300 and pressed Enter.
3. The user entered 150 and pressed Enter.
4. The program uses this data to draw a circle.
Example

• A complete program that gets data from the user and uses that data to draw a circle

• `atoi` and `atof` functions
  - Ignore any spaces that might appear at the beginning of the string
  - Perform the conversion process until an invalid character is encountered
  - Return 0, if the string
    - Is empty
    - Cannot be converted to a number

Notes on Strings

• Numeric values need to be converted to strings before they can be displayed with
  - `dbPrint`
  - `dbText`
  - `dbCenterText`

• The `dbStr` function can be used to convert numeric values to strings

  example:
  ```
  int score = 550;
  dbPrint( dbStr(score) );
  ```
Concept:
The Dark GDK uses the RGB color system to generate colors. In the RGB system, you define a color by specifying values for its red, green, and blue components.

RGB Color System
- 3 color channels:
  - Red
  - Green
  - Blue
- value from 0 to 255
  - 0 is minimum brightness
  - 255 is maximum brightness
- Stored as DWORD
- 8 bits
- 32 bits
Storing Color values in memory

- The Dark GDK library provides a function named `dbRGB` that returns the DWORD value of an RGB color.
  
  \[
  \text{dbRGB}(\text{red}, \text{green}, \text{blue}) ;
  \]

- Example:
  
  \[
  \text{DWORD brightRed;}
  \text{brightRed = dbRGB(255, 0, 0);} \]

Drawing in color

- Example:
  
  - The function `dbInk` changes the current drawing colors
    
    \[
    \text{dbInk(\text{foreground, background});}
    \]

  - All subsequent shapes and text will be drawn in the `foreground` color

  - The `background` color
    
    - Only applies to text
      
      - after calling the `dbSetTextOpaque` function
    
    - Has no effect on shapes
      
      - For drawing shapes, you can pass any color you like for the `background`
Example

how to draw a blue circle:

```c
// Declare variables for the colors blue and black.
DWORD blue = dbRGB(0, 0, 255);
DWORD black = dbRGB(0, 0, 0);

// Change the current drawing colors.
dbInk(blue, black);

// Draw a circle at (100, 100) with a radius of 50.
dbCircle(100, 100, 50);
```

---

Clearing screen

- The Dark GDK library provides two functions for clearing the window
  - `dbClear`
    ```c
dbClear(red, green, blue);
```
    - Clears the window to a specific background color
    - Accepts arguments for the background color’s red, green, and blue components
  - `dbCLS`
    ```c
dbCLS(); // Clears the window to black
```
    - Clears the window to a specific color when you pass a DWORD argument
    ```c
    DWORD blue = dbRGB(0, 0, 255);
    dbCLS(blue);
    ```
Games and Animation

How it works??

The Game Loop

Concept:
The game loop is a special loop used in games and animation programs. It synchronizes the refreshing of the screen with the program's other operations.
The Game Loop

- Virtually all games and animation programs have a loop of some sort that continuously performs operations such as:
  - Calculations
  - Gathering input
  - Moving objects on the screen
  - Playing sounds
  - And so forth

- Hence, the loop must allow the screen to be updated at the appropriate time
  - you must **synchronize** the loop with the updating of the screen

The following functions can be used for synchronizing the loop with the updating of the screen:

- `dbSyncOn`—This function tells the Dark GDK that we want our program to handle the updating of the screen. As a result, the Dark GDK will not attempt to update the screen until we tell it to.
- `dbSyncRate`—This function accepts an argument that specifies the maximum times per second that the screen should be updated. This value is commonly referred to as the *frame rate* or the *refresh rate*.
- `LoopGDK`—This function is used to control the number of times that a loop executes per second. It also tells us, via its return value, whether the user has attempted to end the program. It returns zero if the user has closed the program's window, or pressed the *Esc* key. Otherwise it returns a nonzero value.
- `dbSync`—This function forces a screen update.
The Game Loop

1. Disable automatic screen refreshing
2. Establish a refresh rate
3. Test and synchronize the game loop
4. Display graphics
5. Refresh the screen

```cpp
1 dbSyncOn();
2 dbSyncRate(MaxRefreshRate);
3 while ( LoopGDK() )
   { Statement;
     Statement;
     etc. }
4 dbSync();
```

**NOTE:** Don’t forget to call the `dbSync` function at the end of the game loop. If you leave out this function call, the system will not update the screen.

Giving Control Back to the Dark GDK

- The Dark GDK library has a function called `dbSyncOff` that causes automatic screen updating to start again
- Not needed very often
- Can be helpful in some situations
- For example:
  - When prompting the user to enter a value during the game loop
    - The prompt won’t display until after the screen is refreshed with the `dbSync` function
  - To display the prompt and resume the game loop:
    - Call `dbSyncOff` just before displaying the prompt
    - Call `dbSyncOn` after the user enters a value
**Concept:**

A simple way to create an animation is to write a game loop that draws a shape at a different location during each iteration. Be sure to clear the screen of anything displayed during the previous iteration, though!

```
// Display the animation.
while ( LoopCycles() )
{
    // Clear the screen.
    dBCLS();

    // Draw the ball at the current location.
    dBCircle(CENTER_X, y, RADIUS);

    // Calculate the Y coordinate for
    // the next iteration.
    if ( y > TOP_Y )
    {
        // Decrease the value of y. This
        // will move the ball up the screen.
        y -= DISTANCE;
    }
    else
    {
        // Move the ball back to the bottom
        // of the screen.
        y = BOTTOM_Y;
    }

    // Refresh the screen.
    dBSYNC();
}
```
Clearing the Screen in the Game Loop

- Always clear the screen before drawing a shape
- Otherwise, all drawings of the shape will appear on the screen

Controlling Objects with the Keyboard

**Concept:**
The Dark GDK provides functions that let you know whether certain keys, such as the arrow keys, spacebar, Enter key, and so forth are being pressed. Many games and animation programs allow the user to control objects on the screen with such keys.
Controlling Objects with the Keyboard

- Games commonly allow the player to use keys on the keyboard to control objects on the screen
- The Dark GDK provides a number of functions for the purpose of detecting whether the user has pressed these keys

**Table 7-1 Key detection functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbUpKey</td>
<td>Returns 1 (true) if the Up arrow key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbDownKey</td>
<td>Returns 1 (true) if the Down arrow key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbLeftKey</td>
<td>Returns 1 (true) if the Left arrow key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbRightKey</td>
<td>Returns 1 (true) if the Right arrow key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbControlKey</td>
<td>Returns 1 (true) if the Ctrl key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbShiftKey</td>
<td>Returns 1 (true) if the Shift key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbReturnKey</td>
<td>Returns 1 (true) if the Enter key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbEscapeKey</td>
<td>Returns 1 (true) if the Esc key is being pressed. Otherwise it returns 0 (false).</td>
</tr>
<tr>
<td>dbSpaceKey</td>
<td>Returns 1 (true) if the spacebar is being pressed. Otherwise it returns 0 (false).</td>
</tr>
</tbody>
</table>

Example:

```c
if ( dbSpaceKey() )
{
    dbPrint("You pressed the spacebar.");
}
```

- If the user is pressing the spacebar
  - The `dbSpaceKey` function returns a value of 1 (true)
  - The message "You pressed the spacebar." is displayed
- If the user is not pressing the spacebar
  - The `dbSpaceKey` function returns a value of 0 (false)
  - The message is not displayed
Controlling Objects with the Keyboard

```cpp
// Program 7-4 (KeyDetection.cpp)
1 // This program demonstrates how to perform
2 // continuous key detection in the game loop.
3 #include "GUIMain.h"
4
5 void DarkRed();
6 {
7    display3(); // Disable auto screen refresh
8    for (int i = 0; i < 50; i++) // Set the maximum screen refresh rate
9     while (LoopWin());
10    // Is the user pressing any of the arrow keys?
11    if (CheckKey())
12        divlist("You pressed the up arrow key.");
13    if (CheckKeyDown())
14        divlist("You pressed the down arrow key.");
15    if (CheckKeyUp())
16        divlist("You pressed the up arrow key.");
17    if (CheckLeftKey())
18        divlist("You pressed the left arrow key.");
19    if (CheckRightKey())
20        divlist("You pressed the right arrow key.");
21
22    // Update the screen.
23    display2();
24 }
```

A message is displayed during each iteration of the game loop for the key the user is pressing.

7.3 Controlling Objects with the Keyboard

**NOTE:** You might be wondering why we used a series of if statements instead of an if-else-if statement. The reason is that the user can press multiple keys at the same time. By having separate if statements, the loop will test each key during an iteration. As a result, we will see messages appear for all of the arrow keys that are being pressed. If we had used an if-else-if statement, we would see only one message appear during an iteration, regardless of the number of arrow keys that the user is pressing.
Letting the User Move an Object

To move a circle with the arrow keys:

Prepare the circle:
- Declare and initialize \( RADIUS \) constant
- Declare and initialize \( x \) and \( y \) coordinate variables with starting values

Prepare the game loop:
- Turn on manual refresh
- Set the maximum refresh rate

Letting the User Move an Object

```c
// Constant for the circle's radius
const int RADIUS = 50;

// Variables for the circle's XY coordinates.
// We initialize these with the coordinates of
// the center of the screen.
int x = dBScreenWidth() / 2;
int y = dBScreenHeight() / 2;

dbSyncOn(); // Disable auto screen refresh
dbSyncRate(60); // Set the maximum screen refresh rate
```

```
// The game loop
while ( LoopGDK() )
{
    // Clear the screen.
    dBCLS();

    // Draw the circle at its current location.
    dBCircle(x, y, RADIUS);

    // If any arrow key is being pressed, then
    // move the circle accordingly.
    if ( dBUpKey() )
    {
        y--;
    }
    if ( dBDownKey() )
    {
        y++;
    }
    if ( dBLeftKey() )
    {
        x--;
    }
    if ( dBRightKey() )
    {
        x++;
    }

    // Update the screen.
    dbSync();
}
```
Performing Other Operations

Increasing and decreasing the radius of a circle

```c
// If the spacebar is being pressed, AND the Ctrl key is NOT being pressed, then increase the circle's radius.
if (dbSpaceKey() && !dbControlKey())
{
    radius++;
}

// If the spacebar is being pressed, AND the Ctrl key is being pressed, then decrease the circle's radius.
if (dbSpaceKey() && dbControlKey())
{
    radius--;
}
```

That's all for today!!

Thanks.....