CO1301: Games Concepts
Lab 9 - Drawing Text & Sprites

1. Introduction

1. In this lab, you will learn to display text and use sprites in your TL-Engine projects.

2. Setting Up

1. Create a new TL-Engine project called Lab9_Text_Project.

2. Create a kManual camera at position (0, 0, 0), then implement camera control using keyboard and mouse as you did in Lab3_CameraControl_Project.

3. Before utilising text in the TL-Engine, you need to load a font. Add the code below in the setup area of your code in order to load a font.

   ```
   IFont* myFont = myEngine->LoadFont( "Comic Sans MS", 36 );
   // Load "Comic Sans MS" font at 36 points
   ```

   As you can see, fonts are loaded in a similar way to meshes. A new data type: IFont is used for the fonts. As with meshes you need both the data type of the variable and a name for the variable.

4. Once a font has been loaded you can draw text with it. However, the text will only be drawn for one frame. In order to keep the text permanent, you must put the code for drawing the text in the game loop

   ```
   myFont->Draw( "Hello World", 200, 100 );
   ```

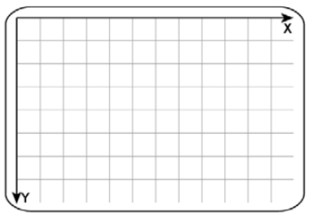
   Adding the code above to your game-loop should display "Hello World" on your screen.

5. You can load any system font e.g. Times New Roman, Adobe Garamond Pro, Comic Sans MS, Monotype Corsiva, stencil std, Charlemagne std etc. Not every font is guaranteed to work correctly. Experiment and see what happens. If none is loaded, the font defaults to 24 point Arial.

6. Now load the cube mesh and create a model from it.

7. Make sure you have implemented camera movement similar to the one in Lab3_CameraControl_Project and use to it to control your camera. Once you have done this move the camera around. What did you notice?

8. Text (and Sprites as you'll see later on) use a different coordinate system to the one used to draw 3D models. Text uses the X and Y axes of the screen. This alternative coordinate system is the same as the one used by the mouse. The screen axes look like this:

   The origin (0, 0) of the coordinate system is at the top left of the screen. Y increases as you go down the screen, whilst x increase as you go to the right.

3. Moving Text

1. Unlike a model, you cannot move text directly. Instead, you will have to keep track of where you would like the text to be drawn.

2. Declare two floating-point variables called fontX and fontY. Set them both initially to a value of 0.

3. Draw the text using fontX and fontY.

   ```
   myFont->Draw( "Hello World", fontX, fontY );
   ```

4. Set up the cursor keys so that they change the values of fontX and fontY within the game loop and so move the text. The cursor keys are: Key_Up, Key_Down, Key_Right and Key-Left. You should then be able to move the location of the text around the screen.

5. There are a number of other things that you can do with the 'Draw'. The Draw method is defined as:

   ```
   Draw( const string& sText, const int iX, const int iY,
       const unsigned int uiColour,
       const EHorizAlignment eHorizAlign, const EVertAlignment eVertAlign )
   ```

   An example of use is:

   ```
   // This will draw red text, centre aligned at the coordinates (100, 300)
   myFont->Draw( "Hello Again", 100, 320, kRed, kCentre );

   //The will draw the phrase in blue in exactly the centre of the screen.
   myFont->Draw( "Hello Again", 640, 320, kBlue, kCentre, kVCentre );
   ```

   Note: that the coordinates used in the last draw method are based on the default window size of Tl-Engine projects (1280, 720). If you are using a different window size, you should take half of your custom dimensions when calculating coordinates for the centre of your screen.

6. The colour of the text is specified in the fourth parameter of the method. The following standard colours are supported:

   - kWhite
   - kGreen
   - kMagenta
   - kDarkGrey
   - kBlack
   - kBlue
   - kYellow
   - kRed
   - kCyan
   - kLightGrey

7. The alignment of the text can be specified in the fifth (horizontal) and sixth (vertical) parameters. Horizontal alignment refers to the horizontal positioning of the text in relation to the coordinates specified. By default, the text starts at the given coordinates (i.e. it is left aligned). However, the text can be centred at the coordinates or right aligned. The constants for horizontal alignment are:

   - kLeft
   - kRight
   - kCentre

   Vertical alignment refers to the vertical positioning of the text in relation to the coordinates specified. The constants for vertical alignment are:

   - kTop

- kVCentre
- kBottom

Using these constants, try to position your text in the top left, top right, bottom left, bottom right and absolute centre of the screen.

4. Displaying More Complex Text

1. Create a new TL-Engine project called Lab9_KeepingScore_Project and copy the code from this file into the project's .cpp file (check the path to the default media folder to avoid errors).

   Run the project and you will see a simple game where you press space bar to shoot a sphere at moving cubes. Get familiar with the gameplay and the code, you will be adding display messages to this game in this section.

2. To output complex text (including variables or functions/methods) you need to be introduced to a new part of C++. You may have used "cout" in this module or in CO1401 in conjunction with the double chevron "<<" to direct text output to the screen, e.g.

   ```
   cout << "hello world";
   ```

   However, it is also possible to use the 'stringstream' type, which can be used like 'cout' to create a string. The stringstream type is used by tL-Engine to display complex text.

3. The steps involved are as follows:

   - Include the sstream library in your program

     ```
     #include <sstream>
     using namespace std;
     ```

   - Create a variable of type stringstream to hold the text you want to draw on the screen. Declare the variable within the game loop, otherwise you'll get problems with repeating text.

     ```
     stringstream outText;
     ```

   - Assign to it the text you wish to display (variables can be assigned), for example:

     ```
     outText << "Sphere Position: " << sphere->GetX();
     ```

   - Use the draw method to draw the text within the stringstream variable, make sure you use the str() command in conjunction with stringstream variable to convert its data into a string of characters to be drawn:

     ```
     myFont ->Draw( outText.str(), 20, 20 );
     outText.str(""); // Clear myStream
     ```

4. Although stringstream gives you access to functionality similar to "cout" do note that outputting newlines (endl or '\n') has no effect in the TL-Engine - use more than one 'Draw' call instead.

5. Back to your Lab9_KeepingScore_project. Your first task is to write code that displays the value of "score", the variable that keeps track of the number of cubes hit by the player. Take a moment to first think about where this code should be placed.

6. Include the stringstream library and remember that you need to use the std namespace as well.

7. Declare an IFont variable called regularFont and load a font of your choice with a font size of 36.

8. Declare a stringstream variable within the game loop and assign to it the value of score (using the format - Score: 0). Then draw it using regularFont at the top right corner of your screen. Make sure you use the appropriate cordinates, align constants, and colour.

9. Currently, the game stops when it enters either gamewon or gameover states. Remove the lines of code that stops the game in each of these states and replace them with code that draws an appropriate message at the center of the screen. Display "Game Over" and the current score when the game is in gameover state, and display "You Won!!!" when the game is in gamewon state.

10. Create a new IFont variable called largeFont and load the same font type as regularFont and a font size of 72. Use largeFont to display the game over and game won messages.

5. Drawing Sprites

1. Sprites are 2D bitmaps. Sprites were once used to draw all of the graphics in games. However, with the introduction of 3D graphics, they are no longer the centre of computer graphics.

2. Whilst models have overtaken sprites as the mainstay of computer graphics, sprites are still extremely important.

    - Sprites commonly used within User Interfaces, Menus and Loading Screens.
    - Sprites can be used to create effects such as smoke, explosions and rain.

    In this section, you will use a sprite as a backdrop for a User Interface.

3. In many ways you will find the sprite operations similar to those of models. However, sprites are simpler in as much as you only have to consider movement in 2D: up and down, or left and right. The sprites do have a Z component, but this is only used to sort the sprites, i.e. to calculate which sprite is in front, and hence visible, if two or more sprites overlap.

4. The naming convention follows the format you are familiar with. Essentially you can replace the word Model with the word Sprite. An important difference is that you do not need to load a Mesh.

    The data type for a sprite pointer is ISprite.

5. Sprites are created using CreateSprite(). Formally CreateSprite is a method of myEngine. There are a set of movement and positioning methods for sprites. These are: GetX, GetY, SetX, SetY, SetZ, SetPosition, MoveX, MoveY, and Move.All pixel coordinates are in *floating point*

    ```
    ISprite* backdrop = myEngine->CreateSprite( "backdrop.jpg", 200, 400 );
    backdrop->moveX( 0.1f );
    ```

6. Create a new project called Lab9_Sprites_Project.

7. Load "BlackBall.jpg" as a sprite backdrop and position it at the bottom of your screen. Then draw the phrase "User Interface" over the sprite backdrop in white.

8. The techniques described above can be used to output real-time information from your game. Load the mesh for a sphere and create one sphere. Set up key controls in order to move the sphere. Use the GetX, GetY and GetZ methods and the text output technique you have just learnt to output the current X, Y, and Z coordinates of the sphere as you move it around. The values should be drawn in the User Interface area.