CS230 Project 2 – Graphics and Physics

Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 4 (Spring Semester) or Week 3 (Summer Semester).

Objectives

The objectives for this Project are four-fold:

- To implement entities using component-based design
- To implement entity construction using deserialization
- To implement functionality for positioning and moving entities.
- To implement functionality for displaying entities.

Description

For this project, you have been provided with a set of header files (.h) that specify the public interface for creating entities and attaching components. You are responsible for creating the associated source files (.c) and implementing the functionality, as outlined in the header files and the lecture notes.

Instructions have been provided below on how to repurpose the two scenes from Project 1 to display moving sprites using two common approaches:

- Textured sprites
- Colored sprites using colored vertices

Game States

In Projects 0 and 1, four levels were implemented as simple scenes. Two of these existing scenes will be repurposed for Project 2. The other two will remain in the project and will be made accessible from the other scenes

As in Project 1, each of the Scene execute functions must append a message to a “Trace.log” file.

- Loading Level 1 must append “Level1: Load”
- Initializing Level 1 must append “Level1: Init”
- Updating Level 1 must append “Level1: Update”
- Rendering Level 1 must append “Level1: Render”
- Exiting Level 1 must append “Level1: Exit”
- Unloading Level 1 must append “Level1: Unload”

CS230 Project 2 – Graphics and Physics

Files

For Project 2 you will need to copy certain files from Project 1. To do so, simply drag-and-drop the files from Project 1 into the source folder for Project 2. However, *do NOT overwrite any existing files in the Project 2 source folder!*

NOTE: You may not change the public interface of the header files (.h) that are provided in Projects 0 through 2, except as expressly directed in the instructions. Should you modify these header files in any way, exercise extreme caution, as adding, removing, or modifying the public interface will result in a penalty to your project grade.

NOTE: The Entity, Mesh, Physics, Sprite, SpriteSource, and Transform structures must all be declared in their associated .c files, not the .h files. Exposing the internal implementation of these modules by declaring the structures in the .h files will result in a penalty to your project grade.

stdafx.h

- ***Important Note:*** The stdafx.h file must be included as the ***first include file in every .c file***. You will encounter build errors if you mistakenly place any other header files before this one.

~~**PlatformSystem.c**~~

- ~~This module has been modified to display a new window title:~~
  - ~~`initInfo.mWindowTitle = "CS230 Project 2 – Graphics and Physics";`~~
- ~~There is no need to make any changes to this file for Project 2.~~

The following header files specify the public interface for a new set of modules. You are responsible for creating the associated source files (.c) and implementing the required functionality, as outlined in the header files, the lecture notes, and the instructions below.

Mesh.h

- This header file declares the public interface for creating and rendering a mesh object using the DigiPen Graphics Library (DGL).
- There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Mesh.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface.

Transform.h

- This header file declares the public interface for storing an entity's position, orientation, and scale within the world space.
- There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Transform.c. You are free to change

the contents of this structure within the .c file as long as you do not change the public interface.
- The contents of the Transform structure may not be accessed directly anywhere outside of Transform.c. The public interface provides everything necessary for this project

Sprite.h
- This header file declares the public interface for a Sprite component.
- There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Sprite.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface.
- The contents of the Sprite structure may not be accessed directly anywhere outside of Sprite.c. The public interface provides everything necessary for this project.
- To successfully draw a Sprite, you will need to perform the following steps:
    - If the Sprite has a sprite source:
        - Set the shader mode to TEXTURE.
        - Call SpriteSourceSetTexture.
        - Call SpriteSourceSetTextureOffset, passing the Sprite's frameIndex.
    - If the Sprite does not have a SpriteSource,
        - Set the shader mode to COLOR.
        - Call DGL_Graphics_SetTexture, passing NULL.
    - Call DGL_Graphics_SetCB_TransformData, passing the translation, scale, and rotation values from the transform.
    - Call DGL_Graphics_SetCB_Alpha, passing the "alpha" value from the sprite
    - Call DGL_Graphics_SetCB_TintColor, passing all 0.0f values
    - Call MeshRender, passing the Mesh from the sprite.
    - Call DGL_Graphics_SetTexture, passing NULL.

SpriteSource.h
- This header file declares the public interface for a sprite source object
- There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into SpriteSource.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface
- The contents of the SpriteSource structure may not be accessed directly anywhere outside of SpriteSource.c. The public interface provides everything necessary for this project

Physics.h
- This header file declares the public interface for handling the motion of an object
- There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Physics.c. You are free to change the

contents of this structure within the .c file as long as you do not change the public interface
- The contents of the Physics structure may not be accessed directly anywhere outside of Physics.c. The public interface provides everything necessary for this project
- You will use the Semi-Implicit Euler integrator to update an object's velocity and position every game loop, as follows:
    - Get the translation from the transform component.
    - Store the translation (as oldTranslation) in the Physics component.
    - Use the Vector2DScaleAdd function to perform the following calculation:
        - velocity = acceleration * dt + velocity
    - Use the Vector2DScaleAdd function to perform the following calculation:
        - translation = velocity * dt + translation
    - Store the new translation in the transform component.
    - NOTE: A grade penalty will be applied if you not use the Vector2DScaleAdd function, as stated above.

Entity.h
- This header file declares the public interface for a simple container storing the individual components associated with a single Entity. Each Entity may contain one each of the following components (more will be added in future projects):
    - Transform
    - Physics
    - Sprite
- There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Entity.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface.
- NOTE: It is possible for an Entity to contain all or none of the specified components. Your code must perform sufficient error checking to ensure that Entities missing one or more components are handled properly (i.e. no crashes, no unexpected side-effects).
- NOTE: It is your responsibility to ensure that all memory allocated for a given Entity is freed when that Entity is destroyed. This includes all components currently attached to the Entity. *Make sure to test your code using the Visual Studio debugger.*
- The EntityRead() function should work as follows:
    - If the Entity and Stream pointers are not NULL,
        - Read a token from the stream
        - Use the token to set the Entity's name
        - While (true)
            - Read a token from the stream
            - If "token" contains "Transform",
                1. Create a new transform component using TransformCreate()
                2. Call TransformRead(), passing the created transform

3. Add the transform to the entity
- Else if “token” contains “Physics”,
  - Repeat steps 1-3 above, replacing “Transform” with “Physics”
- Else if “token” contains “Sprite”,
  - Repeat steps 1-3 above, replacing “Transform” with “Sprite”
- Else if “token” is empty (zero-length string),
  - Break out of the while-loop

EntityFactory.h

- This header file declares the public interface for building a new Entity using data that is read from a file
- The EntityFactoryBuild() function should work as follows:
  - If the filename pointer is not NULL,
    - Open the file using StreamOpen()
    - If the stream was opened successfully,
      - Read the first token from the file using StreamReadToken()
      - Verify that the first token is “Entity” using strncmp()
        - Hint: Use _countof(“Entity”) for _MaxCount
      - If the first token is “Entity”,
        - Create a new entity using EntityCreate()
        - Call EntityRead(), passing the created Entity
        - Close the file using StreamClose()
        - Return the created entity
      - Close the file using StreamClose()
  - Return NULL

The following modules were created as part of Project 1 and will need to be modified for Project 2:

Stream.c/.h

- In Project 1, this module was created to read data from a serial stream (AKA “deserialization”). This module will be further expanded, as per the new function declaration in the Stream.h file and with the addition of a single private variable:
  - Add a new private array for storing “tokens” (single words).
    - `static char tokenBuffer[1024];`
    - This buffer has an arbitrary length that is more than sufficient for the needs of any CS230 project. The correct use of fscanf_s() will help ensure that no buffer overruns occur should the source data ever become corrupted

        - HINT: It is highly recommended that you use the _countof macro to determine the size of a buffer when accessing it. Avoid the use of magic numbers!

DemoScene.c/.h

- In Project 0, these files were modified to incorporate the CS230 Demo functionality. In Project 2, this scene will be made accessible from Level 1 and Level 2.
- You must make the following changes to this file for Project 2:
    - DemoSceneUpdate:
        - If the user *triggers* the '1' key, change the scene to Level1.
            - HINT: Use `'1'`, not `VK_NUMPAD1`!
        - If the user *triggers* the '2' key, change the scene to Level2.
        - If the user *triggers* the '9' key, change the scene to Sandbox.
        - If the user *triggers* the '0' key, restart the current level.
    - DemoSceneRender:
        - ***Due to differences between the Alpha Engine and the DigiPen Graphics Library, a fatal exception can occur unless you add the highlighted line of code when displaying a simple, colored mesh:***
            - `// TODO: Display a simple, colored mesh.`
            - `DGL_Graphics_SetShaderMode(DGL_SM_COLOR);`
            - `DGL_Graphics_SetTexture(NULL);`

SandboxScene.c/.h

- In Project 1, these files were created to test the Vector2D functionality. In Project 2, this scene will be made accessible from the other three scenes.
- There is no need to make any changes to this file for Project 2.

Level1Scene.c/.h

- In Project 1, these files were created to implement a simple scene. The existing functionality will be repurposed for Project 2.
- You must make the following changes to this file for Project 2:
    - Private Structures:
        - Add variables of the following types to the Level1Scene structure:
            - `Mesh*`
            - `SpriteSource*`
            - `Entity*`
    - Private Constants
        - Add the following constants:
            - `static const float groundHeight = -150.0f;`
            - `static const float moveVelocity = 500.0f;`
            - `static const float jumpVelocity = 1000.0f;`
            - `static const Vector2D gravityNormal = { 0.0f, -1500.0f };`
            - `static const Vector2D gravityNone = { 0.0f, 0.0f };`
    - Level1SceneLoad:

        - Read the initial value of "numLives" from a file named "Data/Level1_Lives.txt" (provided).
        - Create a quad mesh with the following parameters:
            - `0.5f, 0.5f, 1.0f, 1.0f, "Mesh1x1"`
        - Create a SpriteSource object.
        - Load a texture into the SpriteSource object with the following parameters:
            - `1, 1, "PlanetTexture.png"`
    - Level1SceneInit:
        - Create a "Planet" Entity by calling EntityFactoryBuild() with the parameter, `"./Data/PlanetJump.txt"`
        - If the entity was created successfully,
            - Get the Entity's sprite.
            - Set the Sprite's mesh and sprite source.
            - Set the Sprite's frame index to 0. While this call is not strictly necessary, it does allow you to test whether the trace message is written properly.
        - Set the background color to white (1,1,1).
        - Set the blend mode to blend.
    - Level1SceneMovementController:
        - Create a new *private* function for moving the "Planet" entity.
            - `static void Level1SceneMovementController(Entity* entity)`
        - Get the Physics and Transform components from the Entity
        - Verify that the pointers are valid
        - Get the current velocity from the Physics component and store it in a local variable. (Hint: you will need to dereference the return value)
        - Check for VK_LEFT and VK_RIGHT key presses, as follows:
            - If VK_LEFT is pressed, set velocity.x = - moveVelocity
            - If VK_RIGHT is pressed, set velocity.x = moveVelocity
            - If neither is pressed, set velocity.x = 0
        - If VK_UP is "*triggered*"
            - Set velocity.y = jumpVelocity
            - Set the physics acceleration = gravityNormal
        - Check for "landing", as follows:
            - Get the Transform component's current translation
            - If Y translation is < groundHeight
                - Set Y translation = groundHeight
                - Set velocity.y = 0
                - Set the physics acceleration = gravityNone
                - Decrement numLives by 1
                - If numLives <= 0, then set next scene to Level2
        - Set the Physics component's new velocity

- Level1SceneUpdate:
  - Remove any existing code and replace it with the following:
  - Update the Planet Entity.
    - Call Level1SceneMovementController().
    - Call EntityUpdate().
  - If the user *triggers* the '1' key, restart the current level.
  - If the user *triggers* the '2' key, change the scene to Level2.
  - If the user *triggers* the '9' key, change the scene to Sandbox.
  - If the user *triggers* the '0' key, change the scene to Demo.
- Level1SceneRender:
  - Call EntityRender().
- Level1SceneExit:
  - Free the Planet Entity using EntityFree.
- Level1SceneUnload:
  - Free the SpriteSource object using SpriteSourceFree.
  - Free the Mesh object.

Level2Scene.c/.h

- In Project 1, these files were created to implement a simple scene. The existing functionality will be repurposed for Project 2.
- You must make the following changes to this file for Project 2:
  - Private Structures:
    - Add variables of the following types to the Level1Scene structure:
      - `Mesh*`
      - `Entity*`
    - The 'numLives' and 'numHealth' variables are no longer used and can be removed.
  - Private Constants
    - Add the following constants:
      - `static const float spaceshipSpeed = 500.0f;`
  - Level2SceneLoad:
    - Call MeshCreateSpaceship to create a "unit"-sized triangular mesh.
  - Level2SceneInit:
    - Create a "Spaceship" Entity by calling EntityFactoryBuild() with the parameter, `"./Data/SpaceshipHoming.txt"`.
    - If the entity was created successfully,
      - Get the Entity's Sprite.
      - Set the Sprite's Mesh.
    - Set the background color to black (0,0,0).
    - Set the blend mode to blend.
  - Level2SceneMovementController:
    - Create a new *private* function for moving the "Spaceship" entity.

- `static void Level2SceneMovementController(Entity* entity)`
  - Get the Physics and Transform components from the Entity.
  - Verify that the pointers are valid.
  - Get the mouse cursor position (in screen coordinates):
    - DGL_Input_GetMousePosition()
  - Convert the screen coordinates to world coordinates:
    - DGL_Camera_ScreenCoordToWorld()
  - Get the spaceship's current translation
  - Calculate a direction vector from the spaceship to the mouse position.
    - Hint: Use Vector2DSub() to subtract one vector from another.
    - Normalize the direction vector using Vector2DNormalize()
  - Set the transform's rotation, using Vector2DToAngleRad() to convert the direction vector into an angle (in radians)
  - Set the Physics component's velocity = direction vector * spaceshipSpeed
- Level2SceneUpdate:
  - Remove any existing code and replace it with the following:
  - Update and display the Spaceship entity:
    - Call Level2SceneMovementController()
    - Call EntityUpdate()
  - If the user *triggers* the 'Z' key, set Spaceship sprite's alpha value = 0.5f
  - If the user *triggers* the 'X' key, set Spaceship sprite's alpha value = 1.0f
  - If the user *triggers* the '1' key, change the scene to Level1
  - If the user *triggers* the '2' key, restart the current level
  - If the user *triggers* the '9' key, change the scene to Sandbox
  - If the user *triggers* the '0' key, change the scene to Demo
- Level2SceneRender:
  - Call EntityRender().
- Level2SceneExit:
  - Free the Spaceship entity using EntityFree.
- Level2SceneUnload:
  - Free the Mesh object using MeshFree.

Submission Requirements

- The project must build cleanly, with no errors or warnings.
- Once the assignment has been completed, create a submission .zip file by performing the following steps:
  - Select the following files and folders:
    - "Assets" folder
    - "Data" folder
    - "DGL" folder
    - "Source" folder
    - Project2.sln

        - Project2.vcxproj
        - Project2.vcxproj.filters
    - Right-click on one of these files and select the option:
        - "Send to" -> "Compressed (zipped) folder"
    - The resultant .zip file **must not** include any of the following Visual Studio generated folders:
        - Folders: .vs, "Debug", "Release", "x64"
    - Rename the resultant .zip file using the following naming convention:
        - CS230S23<section letter>_<Login ID>_Project2.zip
            - Example: CS230S23A_john.doe_Project2.zip
- Upload the submission .zip file via the Moodle page for your CS230 section (A or B)
- It is your responsibility to ensure that the project was submitted properly. Once the submission has been uploaded, it is ***highly recommended*** that you verify that the submission process was completed successfully by performing the following steps:
    - Return to the home Moodle page for your section (A or B)
    - Click on the assignment submission link
    - Download the .zip file to your computer
    - Unzip the contents of the .zip file into an empty folder
    - Open the Visual Studio solution file
    - Clean and rebuild the project
    - Verify that the program runs correctly (within Visual Studio is fine)

Assignment Grading Guidelines

- A -25% penalty will be applied for each week or portion of a week that the project is submitted late.
- A -10% penalty will be applied to any submissions that are performed incorrectly (e.g. incorrect .zip format, submitting extraneous files, etc.)
- A -10% penalty will be applied to any submissions that do not conform to the naming convention specified in the Submission Requirements section.