Game Graphics

Many game engines supports multiple graphics APIs, eg.: DirectX, OpenGL, Vulkan, etc. It's a must for programmers to make their engine being available to switch library easily.

In this task we will try to achieve this functionality by writing sketch of such a system for game engine.

Project includes:

1. Program.cs containing main function.
2. Graphics APIs' objects interfaces.

Assumptions:

1. Adding new platgraphics APIform should not cause any change in game code (and in other graphics APIs code).
2. Only one graphics API will be used through entire game code.
3. Game code should not be aware of used graphics API.
4. Remember that required implementation is similar for each graphics API, but in real scenario underlying logic can be completely unrelated. [This is important].
5. Provided files cannot be modified.

Your task:

1. Implementation of Graphics APIs objects for 3 platforms: DirectX, OpenGL, Vulkan. Description below.
2. Implementation of function that will behave as game. This part will work as game code. It will be invoked for each platform. Description below.
3. Main function implementation, which will run created function for all available platforms.

Game function:

This function will emulate action of real game. Create some objects, invoke some methods and then ends. This function should NOT be aware of selected graphics API.

What should be done inside:

- Create IRenderer object.
- Create 2 I3DModel objects and set their positions.
- Create 2 Textures and set their files (distinct file names, but don't have to be real, eg. "DistinctFileName1" and "SuperFileName2").
- Set one model and texture in renderer.

- IRenderer.Render().
- Set not used before model and texture in renderer.
- IRenderer.Render().

Main:

Run game function for each available platform.

Interfaces Implementation:

DirectX:
- I3DModel: Postion setter remembers position, and SetActive writes: “DirectX model at position X,Y,Z set active”, where X,Y,Z is set position.
- ITexture: Content setter remembers texture file name. SetActive writes: “DirectX texture from file FILENAME activated”.
- IRenderer: Texture/Model setter remembers given object. Render writes “Before activating”, next invokes SetActive of each one and writes: “DirectX is rendering model…”.

OpenGL:
- I3DModel: Postion setter remembers position, but Y is multiplied by -1. SetActive writes: “OpenGL model at position X,Y,Z set active”.
- ITexture: Content setterappends “.superFormat” to fileName and remembers it. SetActive writes: “OpenGL texture from file FILENAME activated”.
- IRenderer: Texture/Model setter remembers given object and immediately activates it. Render only writes “OpenGL is rendering model…”.

Vulkan:
- I3DModel: Postion setter remembers position with X and Z coordinates swapped, and SetActive writes “Vulkan model at position X,Y,Z set active”.
- ITexture: Content setterremembers reversed filename. SetActive writes “Vulkan texture from file FILENAME activated”.
- IRenderer: Texture/Model setter remembers given objects. Render writes “Before activating” then activates all objects and writes “Vulkan is rendering model…”