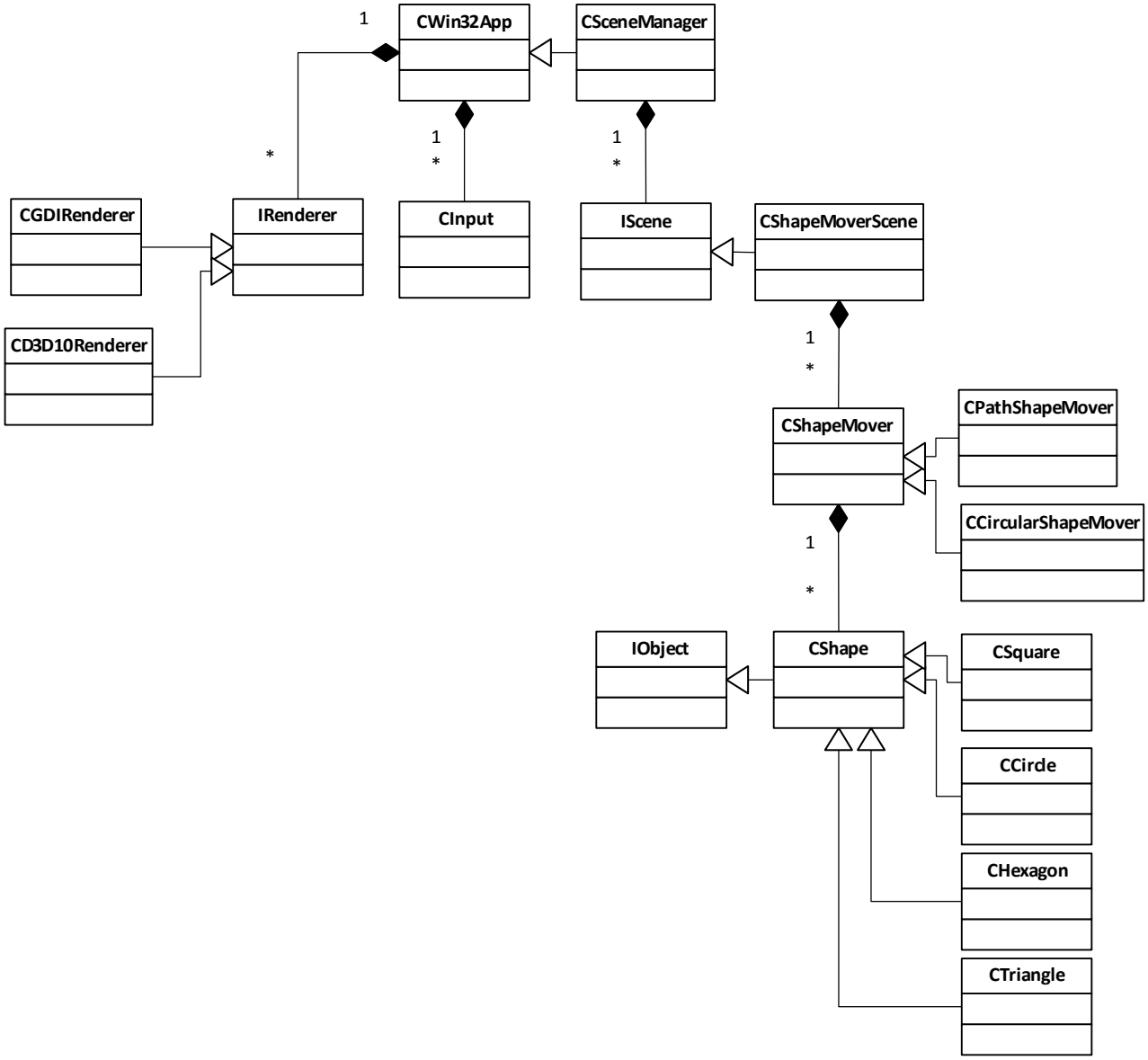
Introduction

With my solution I tried to create something that would prove that I could write reusable and extensible code. I don't have much experience with templates outside of using them for containers so I avoided that entirely.

High Level UML

The following UML illustrates the architecture of the solution at a high level. There are some additional classes such as the INI parser that isn't shown here.



Shapes

I treated shapes as sprites, ie they are simply renderered and contain no/little logic. I created another class called CShapeMover to move the shapes. The ShapeMover can be thought of as an "Actor" or something that will do something inside the scene. This approach of separating the sprite and the actor means that the visuals of the shape and the movement of the shape can be treated separately.

Data Parsing and Scene Loading

I used an existing INI Parser from a school project to load in scenes (ie shape data). The format of the data can be viewed inside one of the INI files inside the bin folder. Two test scenes have been provided.

Note: Currently the code does not check or handle bad data. Could break if values are missing – I didn't test it.

Renderer

I used a simple GDI renderer for a lot of my school projects. For this project I cleaned existing code, added an abstraction layer ie. IRenderer, and a DirectX10 Renderer. The renderers can be switched using the buttons inside the application. I had trouble filling in the area inside polygons so I've only rendered the outlines using the line list topology.

Scene Management

The CSceneManager contains the main loop. It is essentially a state machine that updates the currently active scene. The scenes have built in button functionality for menus. I like this approach because it allows me to easily create additional scenes to quickly test different things within the same project.