Input Support

Lucid3D.System Design Specification

# Overview

## Customer Scenarios

The key scenarios to deliver in the input portion of the system layer are:

1. Consistent, direct access to input state
2. Streamlined state for digital inputs (button up/down, just pressed, just released, etc…)
3. Support for keyboard, mouse, gamepad, and touch

## Goals

1. Lightweight to use. Should be fairly non-intrusive to the existing code base if being used for support functionality.
2. Intuitive and easy to find functionality in the component.
3. Very high performance. In many cases, these pieces of functionality will be the foundation of games, and must allow that game to reach its maximum potential.

## Non-Goals

1. Completely hide the nature of the input device.
2. Be a semantic-based, device-independent, event driven input system (see Lucid3D.Engine or Lucid3D.Framework for further abstractions above what’s provided here).

# Features

To achieve the goals and scenarios for this component, the work will be divided into the following features. It is intentional that these line up almost exactly with the customer scenarios:

1. A general interface to access the various types of devices.
2. State objects for each type of input, which capture a snapshot of the state at that interval (per frame).
3. Rich context in the state objects, such as internally tracking the previous state, to expose differential state (just pressed, just released, holding, pressing harder, pressing lighter, etc…)

# Detailed Design

## Basic INPUT objects and interfaces

The primary goal of the Lucid3D.System input system is to provide easy access to input from various devices, including keyboard, mouse, gamepads, and touch. Each device has its own nuances and specifics, but ultimately each device provides data that falls into one or more of the following buckets:

1. Digital State: Pressed, Not Pressed (ex: keyboard keys, gamepad buttons, mouse buttons, etc…)
2. Analog State: Normalized (0, 1) (ex: gamepad triggers, thumbstick, joystick, etc…)
3. Positional State: Screen Point (ex: mouse pointer, touch point)

By tracking these states, and comparing the current state with the previous state, we can obtain second order differential states, such as:

1. Digital Difference: Just Pressed, Just Released, Still Pressed (Held)
2. Analog Difference: Increased, Decreased
3. Positional Difference: Move
4. Combination of Differences: Click & Drag, Gestures

The caller is required to call Update() on the input interfaces as often as they’d like to refresh the state. Normally, this will be done once per frame.

Lucid3D provides the following core types to expose the states discussed:

\_\_interface IKeyboard

{

void Update();

bool IsDown ( \_\_in byte virtualKey ) const;

bool IsHeld ( \_\_in byte virtualKey ) const;

bool JustPressed ( \_\_in byte virtualKey ) const;

bool JustReleased( \_\_in byte virtualKey ) const;

};

\_\_interface IMouse

{

void Update();

POINT GetPosition () const;

POINT GetMovement () const;

bool IsDown ( \_\_in byte button ) const;

bool IsHeld ( \_\_in byte button ) const;

bool JustPressed ( \_\_in byte button ) const;

bool JustReleased( \_\_in byte button ) const;

};

namespace Input

{

HRESULT GetKeyboard ( \_\_deref\_out KeyboardPtr\* ppKeyboard );

HRESULT GetMouse ( \_\_deref\_out MousePtr\* ppMouse );

}