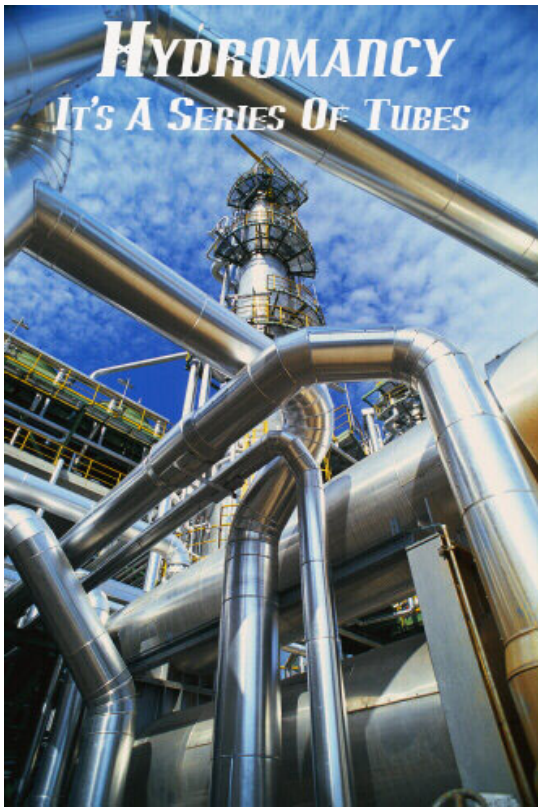
1 Basics

This section lays out the basics of our project.

Title: Hydromancy

Tagline: This game is a series of tubes

Team: Joshua Dawidowicz (jdawidow), Jesse Errico (jerrico), Jacob Frank (jfrank), Michael Feldman (mcfeldma), Micah Lapping-Carr (mlapping)



Core ideas/concepts: In Hydromancy, the player is presented with a series of 3D puzzle levels in which he must eliminate all of the fire in the level before continuing. In doing so, the player must make use of a number of tools, objects, and fluids. Each fluid is simulated at an extremely realistic level; each obeys Navier-Stokes equations for fluid dynamics and Snell's law for light refraction. The simulation should be as close to actual physics as possible, and at the very least observe the law of least astonishment. To do so, we will use an external library which employs computational fluid dynamics (CFD) rather than more simple particle-based simulations. Further, the user may interact with the environment using a mouse and keyboard or Wiimote and Nunchuk; the Wiimote/Nunchuk combination will employ the Wiimote's infrared sensor and both controllers' accelerometers. A wide variety of fluids will be used, including water and other standard Newtonian fluids and non-Newtonian fluids such as a water/cornstarch mix; gasses may be included as well. Fluids will interact fully with the environment as governed by realistic laws of physics and the law of least astonishment, and will also be guided by additional characteristics such as flammability to allow further interaction with the environment.

2 Statement of Requirements

This section describes the requirements for the project independent of implementation concerns.

The following are basic requirements for Hydromancy:

- A simple user interface for both mouse/keyboard and Wiimote/Nunchuk to manipulate objects and fluids in 3D
- Identify when a level has been completed or failed, notify the user, allocate score, and load the subsequent level (in the case of a failed level, reload the current level). Levels are always running (but can be reset); there is no "paused" mode
- A level editor using controls similar to those found in the normal game mode, allowing the user to save and later play through player-created content
- Average a framerate of above 15 FPS on MSLab computers
- Audible audio cues for object manipulation (click sounds on mouseover, select, drag, drop, menu items; contextual sounds for interactive objects)
- Compatible with the following hardware: Wiimote and Nunchuk, Bluetooth reciever
- A pause menu accesible during gameplay with the ability to restart a level, quit a level to the main menu, and change user options (see below)
- A main menu accesible at startup and after quitting a level with the ability to start a selected level, create a new level, change user options (see below), exit the game, or view credits
- Configurable user options including sound, graphics, and input configurations

The following are features which may be added given sufficient time but are not required:

- Tutorial level that explains gameplay and user interactions with voiceover and visual cues (e.g. player loses control of mouse and watches a scripted AI solve the tutorial level, with voiceover)

3 Design

This section lays out a plan for the architecture of the project independent of technical considerations.

Our overall design will have the World at the top level. This class will handle all of startup functionality, such as initializing all the libraries, parsing a level's XML with the **XML Manager**, creating the objects of the level from factory methods, updating the game state with the **Physics** class while the game itself is running, and rendering this state. The World class will also manage the input from the user using an InputHandler.

The **XML Manager** will use a method to load a level into the World, and a method to save a level to disk. The details within the XML file will be the models and their locations/orientations, the graphics and textures required, the background music and sound effects, and other metadata.

The **Physics** module will intermittently update the physics of every object in the scene and update graphical information; during rendering, object position information is read from the physics data to update the graphical nodes in Irrlicht.

The objects in the scene will be derived from either **Solids** or **Fluids**. Solids will have methods to get and update their rotational and translational matrices. Also, Solids will need a method to perform the various interactions necessary for the user, such as rotating, translating in the film plane, and translating along the Z axis. Each Solid will also keep track of its corresponding model and mesh. Fluids are more complicated: they must have a variety of parameters, set at creation time but still modifiable, and a method

to get the current mesh representation, for rendering. Their dynamics will be controlled by a customized fluid dynamics library (below). Interactive solids also have additional characteristics that can be enabled by the user (e.g. a fan that can be turned on and off).

User input is handled with Irrlicht input listeners (mouse and keyboard) and a custom Wiimote listener, both of which feed data to a wrapper class to simplify interactions from the object/fluid end. These interactions will include selection of tools, rotation/translation of tools, manipulation of physics, and interaction with the GUI and menus.

The **fluid dynamics library** will make a series of particles behave as though they are simply molecules in a larger fluid. Our library will take into account density, viscosity, velocity fields, and interaction with other fluids and objects. We will accomplish this using three-dimensional matrices in which each cell represents a particle and each particle acts upon its neighboring ones (subject to change; see below).

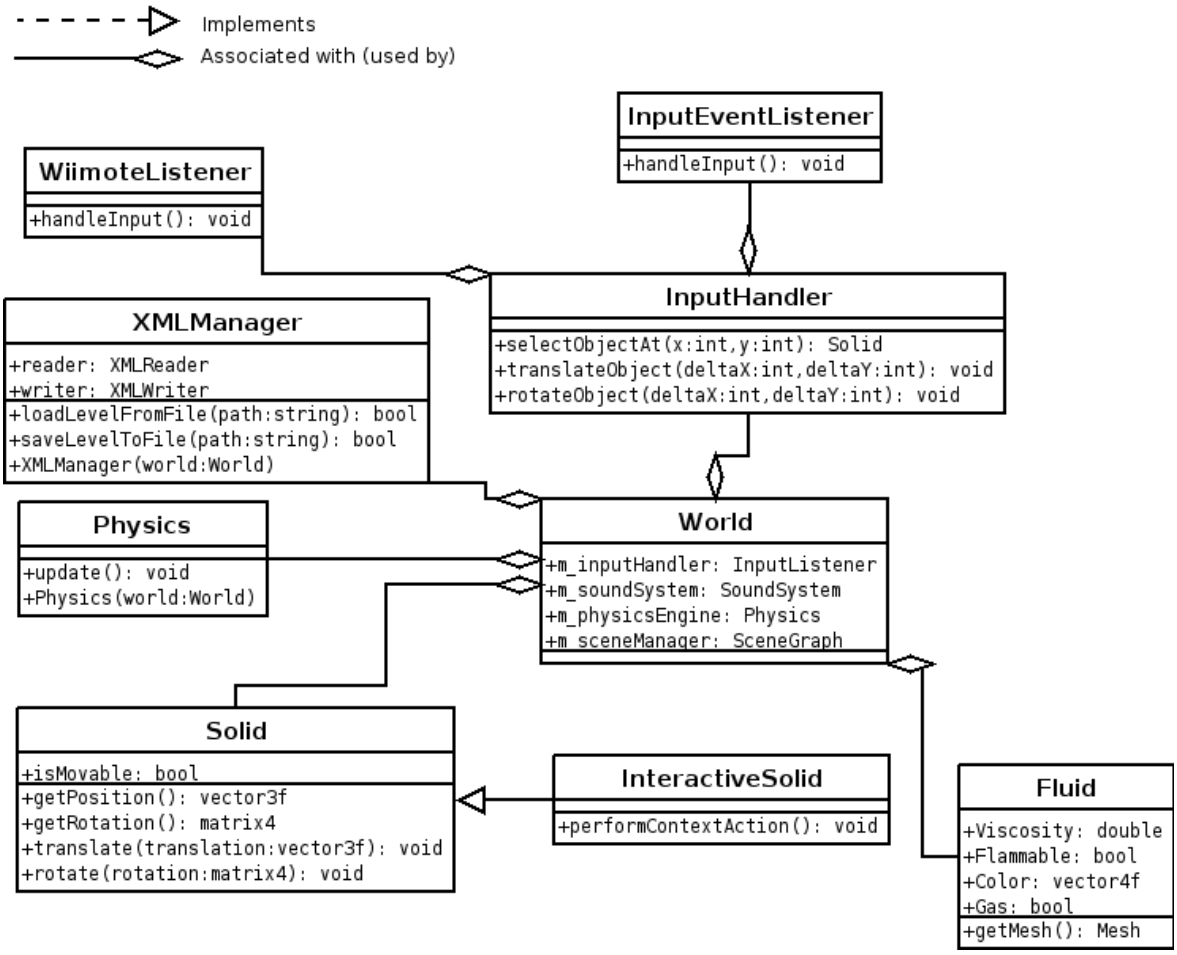
4 Implementation

This section adds technical details to the design and covers code architecture.

System basics:

- Language: C++
- Development tools: Visual Studio 2005, MilkShape for 3D models (converted to .3ds format for compatibility)
- Operating System: Windows XP and Vista
- Engine: Irrlicht
- Physics Library: AGEIA PhysX SDK. This will be used for rigid-body dynamics.
- Fluid Dynamics: We will implement our fluids using AGEIA's smoothed particle hydrodynamics engine (which does not do Navier-Stokes computational fluid dynamics). We will also research an implementation of Jos Stam's 2003 GDC paper (and possibly a Eurographics Workshop paper *Realistic Water Volumes in Real Time* or Mick West's *Practical Fluid Dynamics* from Game Developer Magazine). If this proves to be feasible in 2D, we will experiment with it in 3D, and if this proves feasible, we will replace AGEIA with our own library.
- Audio Library: FMOD
- Wiimote Library: Wii Yourself (Windows Wiimote API - http://wiiyourself.gl.tter.org/)

4.1 UML Diagram



4.2 Explanation of Architecture

This section explains the implementation as shown in the UML diagram above.

- **World:** Central class of the program. The World is the first class that is instantiated and is responsible for initializing all of the other classes. Once a level is loaded through the XMLManager, the World runs the drawing and updating loop until the level is complete or the program is closed. The World will contain instances of classes from libraries including an IrrlichtDevice, an ISceneManager, an IVideoDriver, an IGUIEnvironment, a FMOD::System, and a NxPhysicsSDK.
- **InputHandler:** Takes input captured from the InputEventListener and WiimoteListener and translates it into changes in the World. The InputHandler will be able to find a Solid given an (x, y) screen location and manipulate that object as the user desires.
- **WiimoteListener:** Captures input from a wiimote and calls appropriate methods in the InputHandler. The WiimoteListener will be a wrapper around Wii Yourself, an external library that is freely available.
- **XMLManager:** Loads levels written in XML to the World. The XMLManager will use the World's factory methods to create Solids, InteractiveSolids, and Fluids, and specify a victory condition. The XMLManager will contain an IXMLReader and an IXMLWriter from Irrlicht.

- **Physics:** Given a list of Solids and Fluids, this will update the World on what the next state is. This class will also include functions for computational fluid dynamics.
- **Solid:** Represents a single solid physics object in the game, such as a length of pipe or a basin. Solids will be generated from either factory methods in the World class, since they only require a list of primitives and material properties to be generated, or from existing models created in Milkshape. These models require more complex structures than what can easily be created using Irrlicht and PhysX primitives, but their corresponding meshes will have to be created at import-time by the engines. Some Solids may be translated and rotated by the user. The Solid will be a subclass of ISceneNode, and contain instances of the classes needed by Ageia's NxPhysicsSDK.
- **InteractiveSolid:** Reperesents a Solid that the user can "turn on" or otherwise interact with outside of translation and rotation. The Interactive Solid will be a subclass of Solid.
- **Fluid:** Represents a fluid in the same way the Solid class represents a solid object. There will be one Fluid per type of fluid in the simulation. For example, all the water in a level will be one instance of Fluid, and all the oil will be another instance. The Fluid will be a subclass of ISceneNode and contain instances of the classes needed by Ageia's NxPhysicsSDK. PhysX provides methods to access a mesh that Irrlicht can render.

5 Division of Labor

This section divides the workload among each team member.

- Graphic art and models (3D models and meshes, background images, textures, GUI, promotional): Josh
- Wiimote, user interaction: Micah
- Sound: Mike
- Widgets (music, GUI miscellania, add-ons, etc.): Josh
- XML and level IO: Jake
- GUI: Mike and Jesse
- Physics: All
- Engine and integration: Mike and Jesse

6 Testing Plan

This section lays out a series of feature, integration, and regression tests to perform over the course of the project.

- Physics:

  Collisions of two or more rigid objects

  Fluid generation and removal

  Fluid flow in free-fall

  Fluid flow over single primitives of various material types

  Mixture of fluids

  Additional properties of fluids (e.g. oil is in fact flammable)

Timestep is sufficiently small for accuracy without slowing the movement down too much

Framerate is within parameters on stress test (50+ objects and multiple fluid bodies on screen at once)

- Graphics

  Graphical representation of fluids and solids matches their physics engine equivalent

  Light is properly refracted/reflected with fluids

  User's cursor is always visible

  Context highlighting takes place and is of the correct object, framerate is within paremeters

- XML I/O:

  Levels and objects are properly saved and loaded without any loss of information

  Invalid files are caught and the GUI reflects this

- Input:

  Mouse, keyboard, and Wiimote fulfill all functionality requirements

  No movement glitches occur

  The object under the cursor is always selected when clicked on

  The user cannot place collidable objects inside of each other

  Object rotation and translation works with no skipping.

7 Project Timeline/Milestones

This section provides a timeline for achieving project milestones and realizing deliverables.

- 3/21 - Prototype due today. Particle fluid dynamics implemented, basic rendering of fluid as particles without textures/refraction, basic mouse and keyboard interaction.
- 4/4 - Computational fluid dynamics functional. Rendering of fluids functional (solids and fluids look and act properly).
- 4/11 - Computational fluid dynamics complete (gases and liquids), Wiimote integrated, user interaction complete (but not yet fine-tuned), sound functional.
- 4/18 - Full support for models, levels, and XML.
- 4/25 - Art week (images, models, textures, levels), resolution of any remaining major issues, code cleanup.
- 5/2 - Debug and wrap up any loose ends.
- 5/6 - Final version and demo.
- 5/12 - Postmortem

8 Evaluation Criteria

This section contains a rubric for evaluation of the finished product.

- **Physics (30%)**
  - AGEIA Library for rigid body physics (5%)
  - CFD and Fluid Rendering (20%)
  - Interaction between the libraries (5%)
- **Control (20%)**
  - Basic Keyboard/Mouse interaction (7%)
  - Wiimote Interaction (13%)
- **Gameplay (25%)**
  - Ease of Use (5%)
  - GUI and Menus (5%)
  - Level Design (8%)
  - Victory Conditions (2%)
  - Level I/O (5%)
- **Graphics & Art (15%)**
  - Models (5%)
  - Game Art (5%)
  - Collision Detection for Custom Meshes (5%)
- **Music and Sound (10%)**