OsgVortex
A tool for Authoring
Real-time Physics Simulations
Anders Backman, VRlab, HPC2N
andersb@cs.umu.se
©Anders Backman, VRlab, HPC2N, Umeå University

What is a simulation?
• Emulate the real world?
• Why use physics?
  - Animations?
• Can we simulate the real world?
  - Number of objects
  - Granularity?
• Start with real-numbers, tweak until it works (works == looks good, feels good) (I feel good).

Simulations – a complicated picture

What is OsgVortex?
• A "link" between graphics and physics.
• Enables the user to create objects that have both visual as well as a "physical" attributes.
• OSG – OpenSceneGraph – handles Visual Attributes
  - Open source scene graph for 3D Graphics
  - Written in "pure C++" - Object Oriented
  - Good performance
  - Good/growing community.
  - Portable (Windows, OSX, Linux, Irix, HP-UX, Solaris)
• Vortex – handles Physical Attributes
  - Rigid body simulation toolkit
  - Critical Mass Labs, Montreal Canada.
  - Physical simulation
  - Written in C
  - Portable (Windows, Linux, Irix)

OsgVortex
• Is a 2-level toolkit
  - Script language (file format)
  - Describes a state in a simulation
  - Create objects, joints, etc, give all objects initial state.
  - C++ API
  - Access the previously created objects and manipulate them.
  - Create new objects/joints during run-time.

Creating a simulation
OsgVortex

Simulation Application
- OsgVortex API
- OpenSceneGraph

- Less code for Simulation Application
- Objects are defined once in a script.
- Easier prototyping
- Performance?
  - Multithreaded

Install and use

Windows
- Setup the environment:
  - Resides under: \dantedante\vrvr
  - Mount a network disk: net use o: \dantedante\vrvr
  - Set the environment variable: set VRLAB_PREFIX=o:
  - Execute the setup script:
    \dantedante\vrvr\osgvortex\setup

Test an application:
- Start Microsoft Developer 6.0
  - test

Build your own application
- cd \VRLAB_PREFIX\samples
- cd temp\samples
- setenv VRLAB_PREFIX ~andersb/vr
- execute \$VRLAB_PREFIX/setup.sh

Linux
- Setup the environment:
  - Resides under: ~andersb/vr
  - Set the environment variable:
    - $VRLAB_PREFIX = ~andersb/vr
  - Execute the setup script:
    - \$VRLAB_PREFIX/setup.sh

Test an application:
- cd \VRLAB_PREFIX/osgvortex
- test

Build your own application
- cd /scratch
- cp -r VRLAB_PREFIX/osgvortex/samples .
- cd samples
- make
- test drive it
  - cd ..
  - ./RUNME
The Viewer

- **Navigation:**
  - Left mouse – Rotation
  - Middle mouse – Pan
  - Right mouse – Zoom

- **Key bindings:**
  - w – Toggles textures on/off (only texture engine 0)
  - r – Reset the simulation
  - q – Quit application
  - t – Toggle frame-rate
  - p – Toggles displaying simulation information.

System description

- **World**
- **Object**
- **Material**
- **Joint**

Taking a step in time

- **Solving the system**
  1. Stepping time into the future
  2. Collision handling
  3. Dynamics (collision response)
  4. Updating graphics with new information

- **Step 1-4 takes** \( t \) seconds to perform.

- **Usually it don’t.**
  - If \( t > t_s \) the system moves in slow motion.
  - If \( t < t_s \) the system goes to fast (Charlie Chaplin movie)

- **\( S \) depends on:**
  - the time for rendering graphics
  - the time for solving the physical simulation.

Overall execution

```
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stepping time into the future</td>
</tr>
<tr>
<td>2</td>
<td>Collision handling</td>
</tr>
<tr>
<td>3</td>
<td>Dynamics (collision response)</td>
</tr>
<tr>
<td>4</td>
<td>Updating graphics with new info</td>
</tr>
</tbody>
</table>
```

Synchronous simulation loop

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP</td>
<td>Reading input from user, network etc.</td>
</tr>
<tr>
<td></td>
<td>Manipulate the scene</td>
</tr>
<tr>
<td></td>
<td>Taking a step in the physical simulation</td>
</tr>
<tr>
<td></td>
<td>Calculating collisions</td>
</tr>
<tr>
<td></td>
<td>Handling the collision</td>
</tr>
<tr>
<td></td>
<td>Calculate the new positions and manipulate the scene</td>
</tr>
</tbody>
</table>
```
Asynchronous simulation

Issues with asynchronous simulation

- **Timing of the Simulation**
  - How do we calculate the size of simulation step?
  - Will it vary?
  - On the upside: better use of the CPU especially with 2 cpus
  - On the downside: complexity of system is much higher
  - Communication/synchronization issues
  - Timer probes (what is not a real-time OS?)

- **Parallelization**
  - Multiple threads: each thread can run independently
  - Static: producer/consumer
  - Dynamic: producer/consumer can change dynamically

- **Interesting timing problems**
  - If rendering of the scene takes a long time compared to the physical simulation time, we get:
    - High frame rate for physics, low for graphics
  - If rendering is easy and the physical simulation is complex, we get:
    - Low frame rate for physics, high for graphics
  - Synchronization is a hard issue when we are working with different time frames.
Timing

- On a large scale problem (400 bodies).
- Timestep is 0.01 (100Hz for real-time).

**Asynchronous**
- 1.8GHz Pentium4:
  - Physics FPS: 30
  - Solving problem: 60
  - Dynamics: 40
  - Updating objects: 30
  - Graphics FPS:
- 2* AMD 2000MP+
  - Physics FPS: 25
  - Solving problem: 25
  - Updating objects: 25
  - Graphics FPS:

**Synchronous**
- 1.8GHz Pentium4:
  - Physics FPS:
  - Solving problem:
  - Dynamics:
  - Graphics FPS:
- 2*AMD 2000MP+
  - Physics FPS:
  - Solving problem:
  - Updating objects:
  - Graphics FPS:

Declarative scripts

- An OsgVortex script is not a language.
- It is parsed using a stack-based method.
- Declarative:
  - Everything is evaluated & lazy evaluation.
- Parsing:
  - It is read from disk and is syntax checked, all keys are inserted into a database.
  - Database is interrogated for keywords such as: Object, World.
- Syntax:
  - The meaning of the variables is intepreted by the OsgVortex parser.

OsgVortex Scripts

- How to learn to use OsgVortex?
- Use the LWMDI method.
  - Learning by watching me doing it.
- Or the even more sofisticated HALATDTFY method.
  - Have a look at the demos & try for yourself.
- Important:
  - Reading the manual for getting a sense of how it works.
  - I won't tell you all the documentation I write to cover whole OsgVortex toolkit.
  - The same goes for OpenSceneGraph.

Syntax - Expressions

- Expressions are really a type of its own.
- It is intepreted as FLOAT, though.
- The value of an expression is calculated once (or more!).
- Predefined functions (see table in UserGuide.pdf).
- Example:
  - timing - something in yellow (right).
- Predefined constants: e, and pi. Don't use these as variables, because then e and pi will be outscoped!

Syntax

- OsgVortex resembles a programming language (sometimes).
- It does not have any imperative stuff such as: loops, if then else, etc.
- It is merely for declaring (creating) a state (variables).
- The meaning of the variables is interpreted by the OsgVortex parser.
Syntax - Data types

- The type of a variable is derived from its data.

- A STRUCTURE is a recursive data-type and can contain any of the data-types

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>1</td>
</tr>
<tr>
<td>FLOAT</td>
<td>1.1</td>
</tr>
<tr>
<td>ARRAY</td>
<td>[2, 3, 4.5]</td>
</tr>
<tr>
<td>STRING</td>
<td>&quot;BubbaGumpShrimp&quot;</td>
</tr>
<tr>
<td>STRUCTURE</td>
<td>another_struct</td>
</tr>
</tbody>
</table>

Variables

- Not an imperative language (for anything else than scalars!)
- Therefore

```c
intrect {
    value 3.1415
}
```

- another_struct
- Won't work! But
- value 3.1415
- another_value
- Will.

Syntax - Structures

- Declared using {}
- Example:

```c
struct {
    value 3.1415
}
```

- In this example struct.value is showing how members of structures can be accessed in expressions. (pi is the constant pi).

Syntax - Scope

- Scope? The life length of a variable.
- Example:

```c
old 98
age old // Age == 98
struct {
    old 31
    ID "kalle"
} anotherStruct {
    age old // anotherStruct.old == 31
}
```

- aSecondStruct {
  age old // aSecondStruct.age == 98
}

Syntax - Preprocessor

- Part of Parser

- Include path
- Includes the text from another file. The path is relative to the current file. Line 1 will include line 3.

- macro name (value)
- Defines a macro with a value (optional). Any occurrence of the macro-name in the file will be replaced with value.

- undef macro name
- An undef macro name will undefine a macro.

- define macro name
- Optional text parsing. The text between a define and a macro (or undef) will only be parsed if macro-name is undefined.

- ifdef macro name
- Optional text parsing. The text between an ifdef and a macro (or undef) will only be parsed if macro-name is defined.

OsgVortex Scripts

- There are 5 important parts of an OSV file:
  - Header
  - World
  - Material
  - Object
  - Joint
# OsgVortex Header v1.0

## Header

- Leave as in the sample files. Nothing more to it.
- Can change in the future, just for version handling, but nevertheless important.

## World

- Specifies some important things about the simulation.

```
World {
  // End of World
}
```

- StepLength 0.01 - The length of the timestep used when solving the physical system.
- AutoDisable 1 - To save CPU time, objects can be put to sleep when not moving.
- Gravity [0 0 -9.81] - Specifies the gravity constant.

## Material

- Creates material to be attached to Objects later on.
- Material specifies the frictional attributes of an object.
- Defines what happens when two objects collides. (not much meaning of colliding one object?)

```
Material {
  // End of // End of Material
}
```

- ID “ball floor” - essential to model.
- Tuple [0 1] - Not fixed when attaching to objects later on.
- Friction 2 - Friction coefficient, higher more friction.
- Restitution 0.3 - Bounciness (1 - no energy is lost during a bounce)

## Notes

- Friction coefficient is not what you expect it to be, the mass is hidden in the constant, so getting an accurate value for the friction is hard.
- Use #define macro to give the Tuple values readable names
- If an object is missing an material it is really slippery!!

## Object

- Can be visual and/or physical or both.
- Visual
  - Its visual appearance.
  - Non predefined Primitive or geometry File loaded from disk.
- Physical
  - Its physical appearance, mass, material, etc...
  - Consists of Collision geometry and dynamic body.
  - Collision geometry
    - Either primitive or geometry loaded from disk
    - Not flat
  - Dynamic body
    - Same as geometry + mass, material, etc.

```
Object {
  // End of Object
}
```

## Object - pure visual

```
Object {
  // End of Object
}
```

- Primitive “box” // Box loaded from box.flt in primitives.
- Geometry // The visual geometry will be a box.
Object - both physical and visual

Object { 
  ID "ball1" // Unique id
  Size [0.5] // This will be visualized as a sphere
  Position [0 0 3] // Initial position
  Dynamic 1 // It's a dynamic object, i.e. it can move around
  VisualAttributes { 
    Geometry { 
      Primitive "sphere" // This will look like a sphere
    }
  }
  PhysicalAttributes { 
    Material 1 // Uses material index 1
    Mass 1 // Has the mass of 1
    LinearVelocity [0 0 0] // Defines the object's initial velocity
    AngularVelocity [0 0 0] // No initial rotation velocity
    Geometry { 
      Primitive "sphere" // Physically also represented as a sphere
    }
  }
}

Joint

- Joints are constraints, i.e. hinder the object to move freely according to some rule.
- Connects two (one for Kinematic) objects.
- Types:
  - Spring
  - CarWheel
  - Prismatic (good for elevators)
  - Kinematic (To move objects using keyboards etc.)
  - Linear1
  - Hinge
  - ...

Joints

Joint { 
  ID "spring2" // Unique id
  Enabled 1 // Is enabled by default
  Position [0 0 BALL2POSITION] // Position of joint...
  Object1 { 
    ID "ball2" // Which object to connect the joint to.
  }
  Object2 { 
    ID "ball1" // Which object to connect the joint the other side to.
  }
  Spring { 
    NaturalLength 0.1 // Rest length for spring
    Stiffness 10 // Stiffness
    Damping 0 // Damping
  }
}

C++ API

- The OsgVortex C++ API wraps Vortex
  - Making it object-oriented
  - Removes some of the complexity of using the toolkit
  - Making it multithreaded
  - Some hooks to OpenSceneGraph

DemoTime!

Anders Backmans
Flying Creatures
Questions??