PERCRO Perceptual Robotics Laboratory

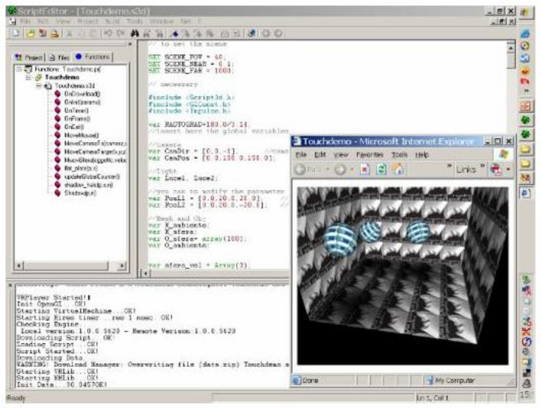
A new Engine for XVR

Daniele Giannetti

Introduction

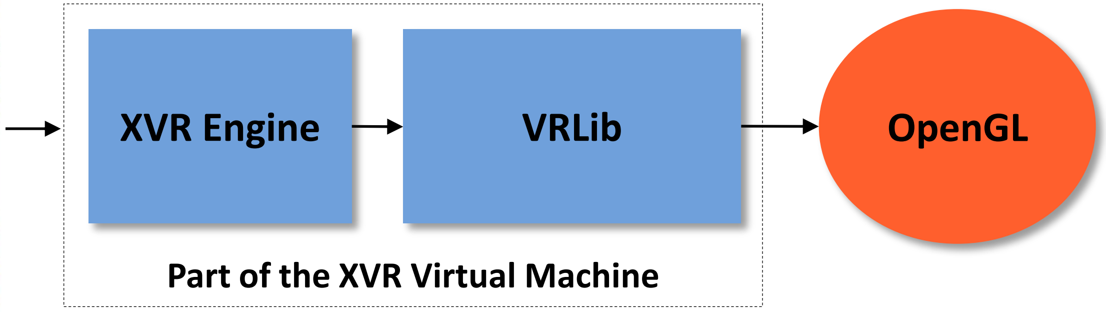
Why a new engine? [1]

- XVR relies on the services of a low-level real-time rendering library in order to generate pictures of the virtual scene during each frame.
- This low-level library is what we call **engine**.
- The engine present in the current version of XVR is called **VRLib** (Virtual Reality Library).
- It is written in pure C++ and uses the OpenGL interface to the graphics hardware.



```
// to set the scene

SET SCENE_FOV = 60;
SET SCENE_NEAR = 0.1;
SET SCENE_FAR = 1000;

// necessary

#include <Script3d.h>
#include <GLConst.h>
#include <Impulse.h>

var RADTOGRAD=180.0/3.14;
//insert here the global variables

//camera
var CamDir = [0.0,-1];        //came
var CamPos = [0.0,150.0,150.0];

//light
var Luce1, Luce2;

//you can to modify the parameter
var PosL1 = [0.0,20.0,20.0];
var PosL2 = [0.0,20.0,-20.0];

//Mesh and Obj
var M_ambiente;
var M_sfera;
var O_sfera= array(100);
var O_ambiente;

var sfera_vel = Array(3);
```

```
VRPlayer Started!!
Init OpenGL...OK!
Starting VirtualMachine...OK!
Starting Hires timer...res 1 msec: OK!
Checking Engine
 Local version 1.0.0.5620 - Remote Version 1.0.0.5620
Downloading Script...OK!
Loading Script...OK!
Script Started...OK!
Downloading Data
WARNING: Download Manager: Overwriting file (data.zip) Touchdemo.s
Starting VRLib...OK!
Starting XMLib...OK!
Init Data...00.04570K!
```



Introduction

Why a new engine? [2]

- The VRLib is quite obsolete (even if powerful and flexible).
- OpenGL and other graphics systems are moving towards a fully programmable approach to graphics (only relying on **shaders**).
- Some common operations are not directly supported by the VRLib (such as shadow casting, which must be realized using XVR shaders and FBOs).
- There is no built-in support for physical simulation of virtual objects (very useful in common applications).

Objectives of the new Engine

Use only modern rendering techniques (shaders) and use a recent version of the OpenGL graphics system.	Introduce a set of easy-to-use, advanced rendering capabilities such as built in support for soft shadow casting.	Introduce physical simulation capabilities in the new engine, in particular we require a full support for rigid bodies simulation.

Introduction

Implementation of the new engine

- A new low-level real-time rendering library was developed in summer and fall 2010, its name is **VR3Lib**.
- It uses version 3.3 of the OpenGL API and has built-in support to rigid body simulation (implemented using the **Nvidia PhysX** solution).
- It can already be used in pure C++.
- It is being integrated in the XVR framework replacing the previous VRLib (work in progress).
- Most of the core features are already available in an internal XVR version (still not available to the community).
- The entire scene management (that happens under the hood from the perspective of the XVR programmer) is done using the new facilities provided by the VR3Lib.
- We are going to show some of the main additional features provided by the VR3Lib and how to use them in XVR.

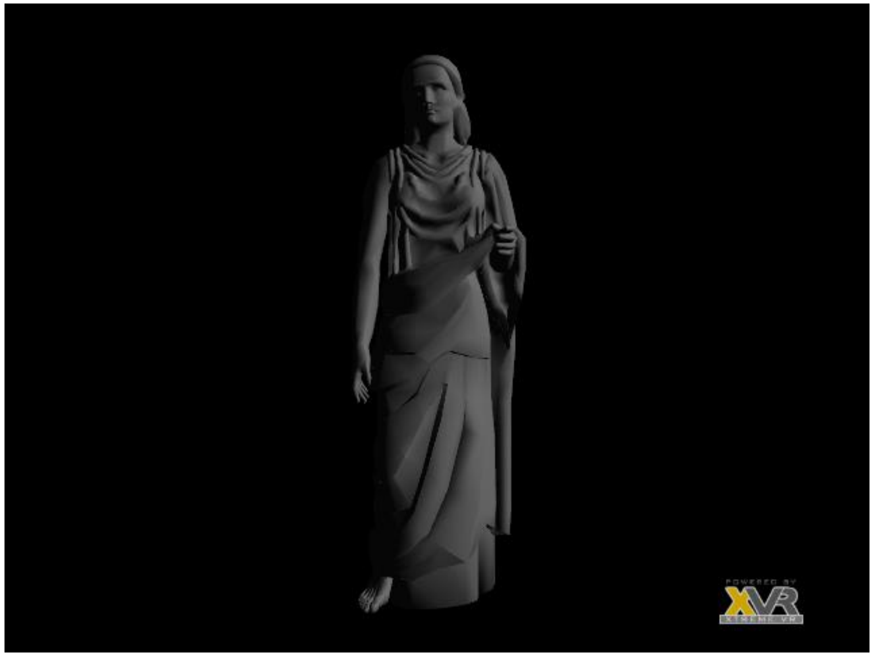
Basic engine capabilities
Basic classes

- Most of the standard XVR classes have been replaced with updated equivalents:
  - CVmCamera → XCamera
  - CVmObj → XObj
  - CVmMesh → XMesh
  - CVmText → XText
  - CVmLight → XLight
- They can be used in a similar way to the previous version, but under the hood the operations are radically different (the VR3Lib has been written from scratch).
- When drawing objects without specifying external shaders, the VR3Lib will use built-in shaders to perform any type of rendering (conforming to the latest versions of the OpenGL Graphics System).

Basic engine capabilities

Example: drawing a simple object

```
// 1_simple
function OnInit(params) {
	obj = XObj("data/Statue.AAM");
	obj.SetPosition(0.0,0.0,-3.0);
}
function OnFrame() {
	SceneBegin();
		obj.Draw();
	SceneEnd();
}
```



- Default direct lighting: weak diffuse lighting with point light at position (40,20,20) – Phong reflection model.
- Even this simple rendering is done with the help of shaders (built-in the VR3Lib engine).

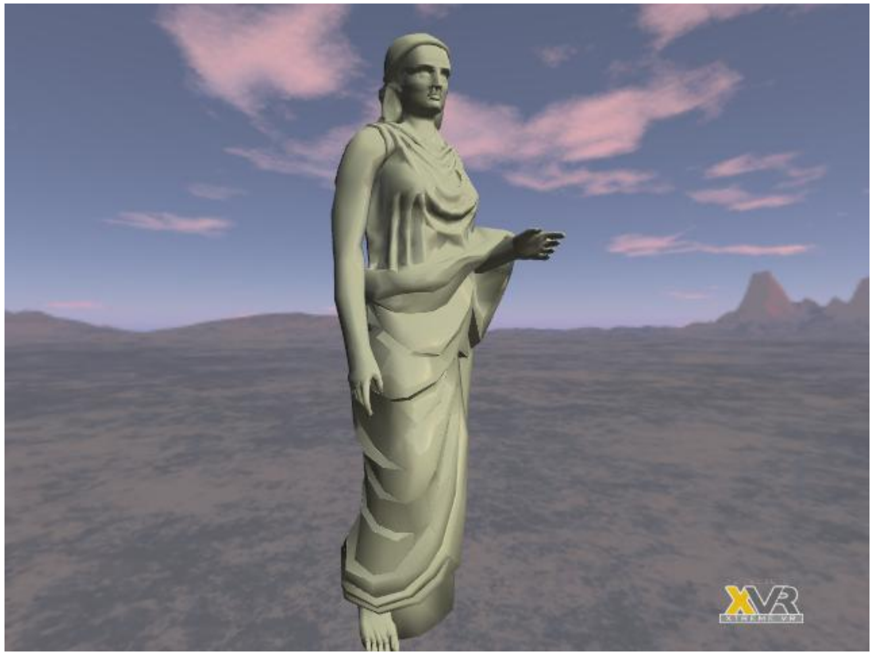
Advanced illumination
Built-in support to IBL

- Direct illumination is a classical but obsolete method to compute illumination of virtual objects.
- The new engine includes built-in **environment mapping** capabilities (a very simple IBL technique).
- Environment mapping is often coupled with **skyboxes**, skyboxes are directly supported by the VR3Lib (cube map format) and can be activated very easily using new XVR functions.
- Real world images can be transformed in cube maps using well-known tools (such as ATI CubeMapGen).
- HDR cube maps are supported.
- A couple of different cube maps are used to compute illumination:
  - Diffuse cube environment map
  - Specular cube environment map

Advanced illumination

Example: Using IBL and skyboxes

```
// 2_IBL_skybox
function OnInit(params) {
	var cam = CameraGetCurrent();
	cam.SetPosition(2.0,0.0,-2.0);
	cam.SetDirection(-1.0,0.0,1.0);
	LoadBackground(
		"data/specular_map.bmp",true);
	SceneEnvironmentMapping(
		"data/diffuse_map.bmp",
		"data/specular_map.bmp");
	obj = XObj("data/Statue.AAM");
	obj.RotateGlobal(180.0, 0.0, 1.0, 0.0);
}
function OnFrame() {
	SceneBegin();
		obj.Draw();
	SceneEnd();
}
```



- The specular map is also used as skybox in this case (common procedure)
- The diffuse map is usually a filtered version of the specular map, using for example a strong blur filter.
- The true flag in LoadBackground() means that the image is a skybox.

Surface details

Normal and displacement mapping

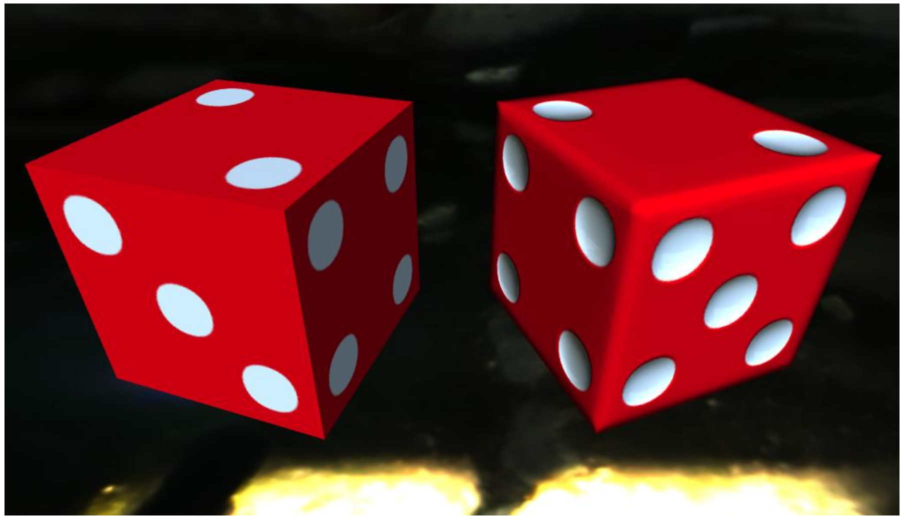
- Highly detailed objects should not be rendered as high-poly models in real time 3d graphics.
- It is desirable to use texture-based methods with low-poly models in order to forge fake surface details.
- The well-known normal mapping technique is available in the new engine, and the proper normal mapping shader is used when a normal map is associated with the loaded model.
- A built-in displacement mapping technique is also available, in this case geometry is altered at render time using dynamic tessellation (implemented in the geometry shading OpenGL stage). The displacement mapping shader is automatically used when a displacement map (or height map) is associated with the loaded object.
- The XVR code to use those functions not different from the previous examples: just load and draw the objects when needed.

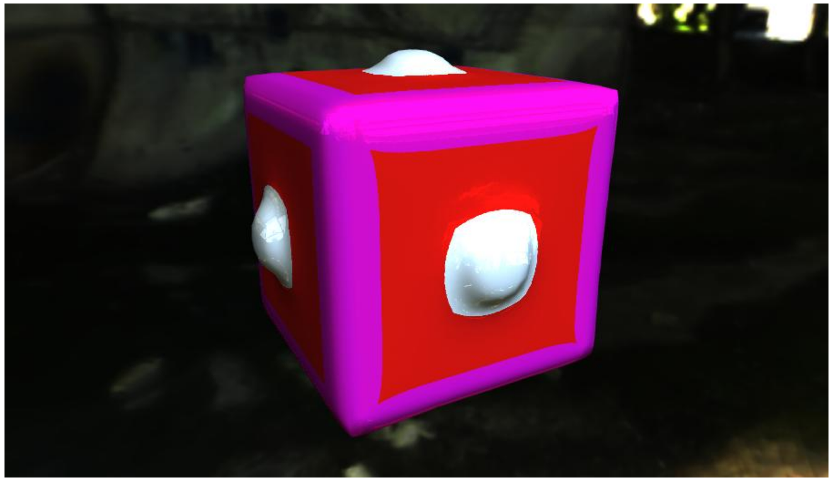
Surface details

Results

Normal mapping

- Both models are low-poly (12 triangles).
- The dice on the right is normal mapped.





Displacement mapping

- The dice is again a low-poly model.
- The bumps are created at render time by displacing vertices obtained from tessellation.

Soft Shadows

Obtaining soft shadows with EVSM [1]

- Many different modern techniques are available to obtain soft shadows (research in this field is very active):
  - VSM (Variance Shadow Mapping)
  - PCSS (Percentage Closer Soft Shadows)
  - ESM (Exponential Shadow Mapping)
  - Penumbra Wedges
  - …
  - Hybrid approaches
- The VR3Lib includes an implementation of the **EVSM** (Exponential Variance Shadow Mapping) method, which improves upon VSM and ESM by solving or reducing common artifacts (such as the well-known light bleeding).
- Algorithm parameters might be tuned to get the best result depending on the particular scene.

Soft Shadows

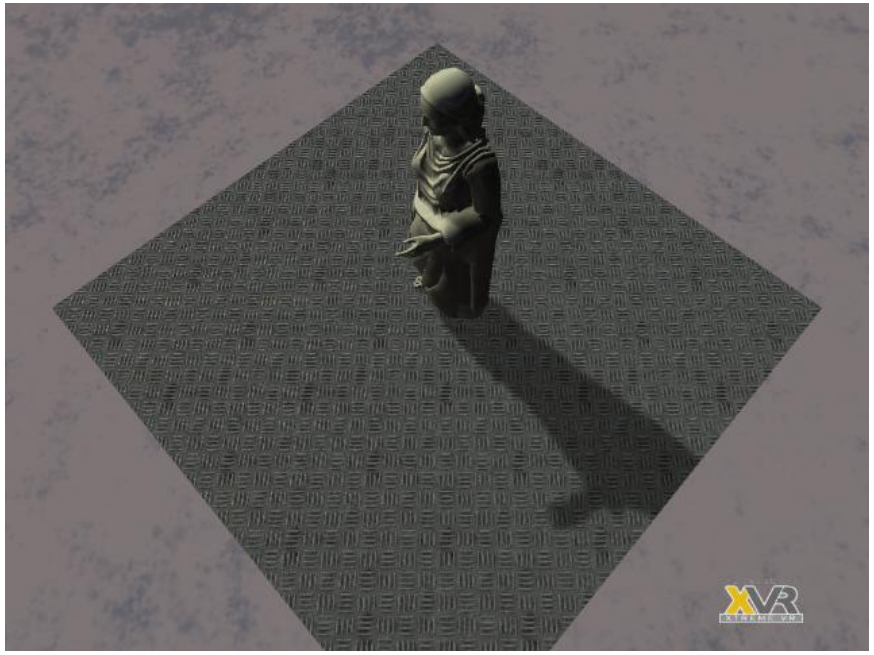
Obtaining soft shadows with EVSM [2]

- EVSM requires two rendering passes, and a filtering pass in the middle
    1. Render the shadow map
    2. Filter the shadow map
    3. Render the scene with shadows
- This is hidden from the XVR user as the first two steps are performed automatically by the VR3Lib.
- The user must simply initialize shadow mapping (using ShadowInit()), specify which objects should receive and cast shadows and create the desired shadow sources (**XShadowSource** class).
- A shadow source is intended in XVR as an entity that causes objects to cast shadows (it might be in the same position of a light, but a light is not required to cast shadows, this is useful when using environment maps).
- EVSM algorithm parameters may be altered on a per-source basis.

Soft Shadows

Example: object casting a shadow

```
// 5_shadows
function OnInit(params) {
	[...]
	ShadowInit();
	ShadowEnable(VR_SHADOW_FACE_CULLING);
	ShadowDisable(VR_SHADOW_VF_CULLING);
	ShadowDisable(VR_SHADOW_SOURCE_CULLING);
	source1 = XShadowSource(
			0.0, 5.0, -5.0, // source position
			0.0, 1.0, 0.0, // up vector
			0.0, -1.5, 0.0, // target
			60.0, // fovy
			1.0,  // aspect ratio
			1.0,  // znear distance
			20.0, // zfar distance
			512,  // map width
			512);  // map height
	source1.Enable();

	obj1 = XObj("data/Statue.AAM");
	obj1.RotateGlobal(-90.0,0.0,1.0,0.0);
	obj2 = XObj("data/Floor.AAM");
	obj2.RotateGlobal(-90.0,1.0,0.0,0.0);
	obj2.SetPosition(0.0,-1.5,0.0);
}
```

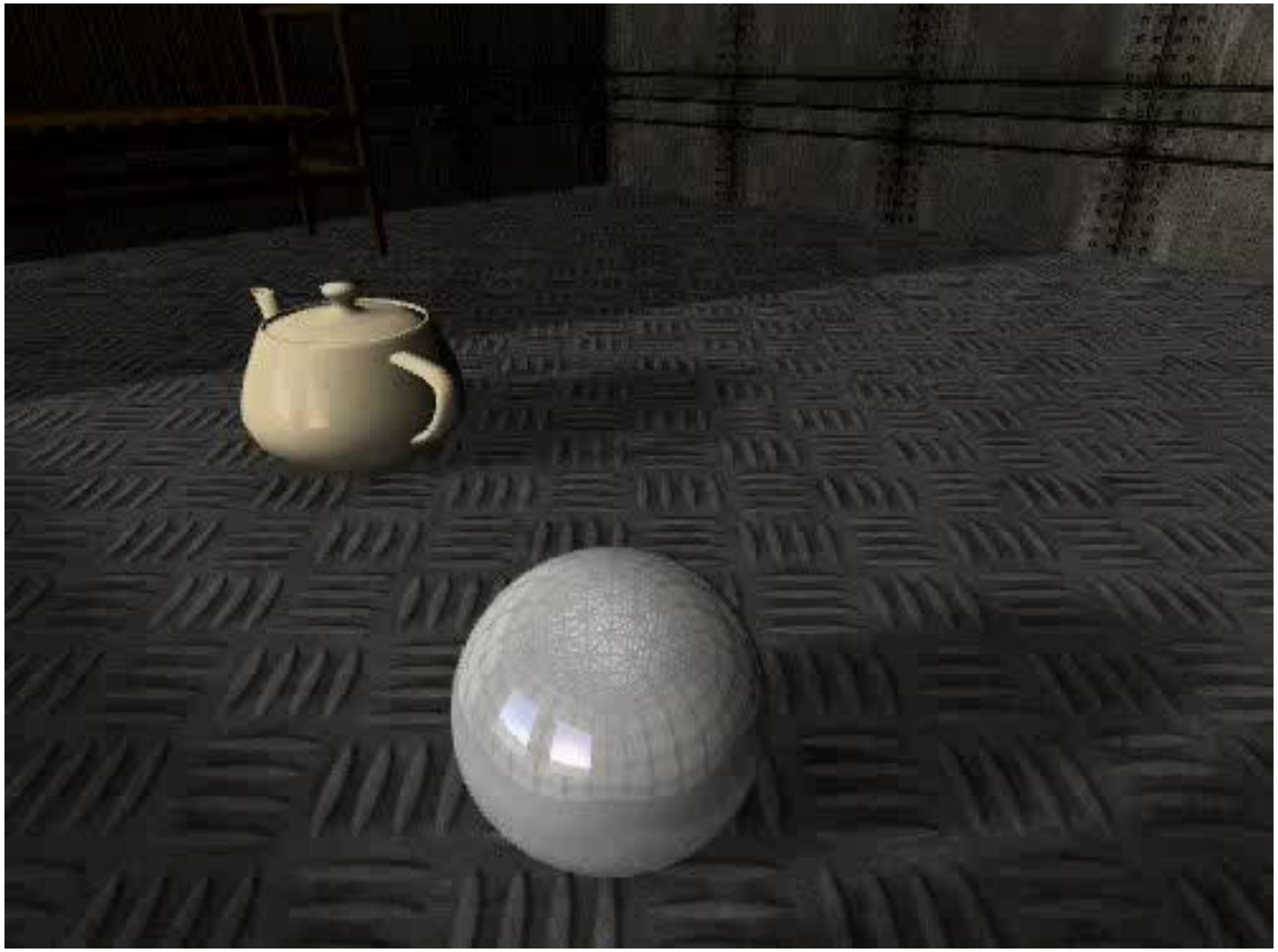


- In this example, all objects are automatically registered as both casters and receivers (ShadowInit() has an optional parameter to specify manual registration).
- The drawing code is the usual one, no new operations are needed.

Soft Shadows
Results

Some well-known artifact may still be visible when using EVSM, but the results are usually satisfying.

EVSM is a fairly innovative technique, relevant literature on it has yet to come.



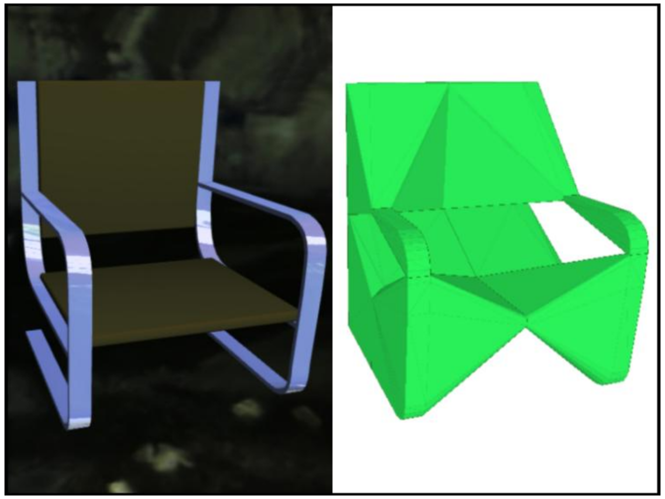
Physical Simulation
Rigid body simulation capabilities [1]

- The Nvidia PhysX engine is used inside the VR3Lib to efficiently obtain position and rotation of simulated rigid bodies.
- This is transparent to the XVR user, who simply initializes physical simulation by calling PhysicsInit().
- A set of functions is available to control the simulation (pause, stop, restart).
- Some physical property of the simulated objects can be changed during simulation.
- Apart from simple rigid body simulation, joints, motorized joints and contact notifications are supported by the VR3Lib (this allows construction of complex kinematic chains).
- Three simulation types for virtual objects:
  – Dynamic: affected by collisions and forces.
  – Kinematic: will collide with dynamic objects, moves only upon user actions.
  – Static: will collide with dynamic objects, never moves.

Physical Simulation

Rigid body simulation capabilities [2]

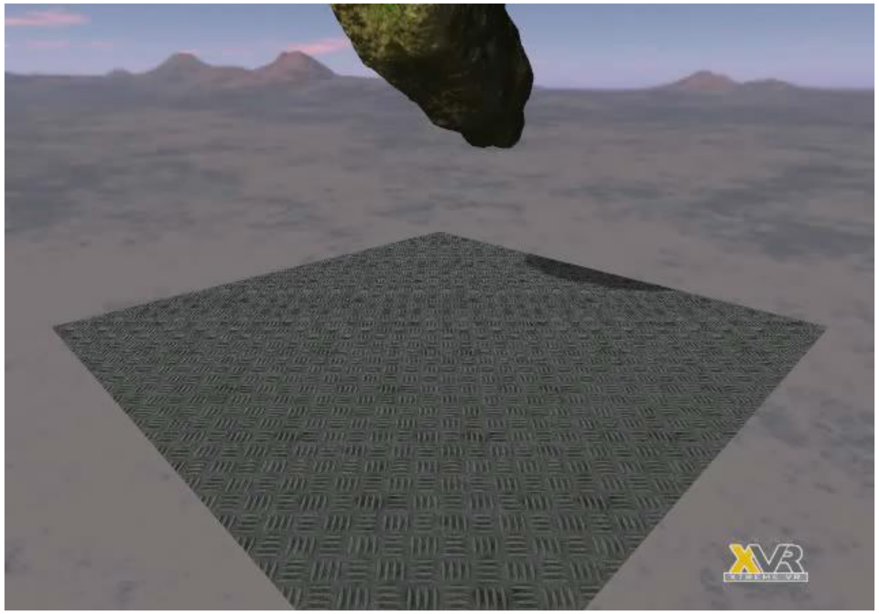
- PhysX is unable to simulate concave meshes for dynamic and kinematic objects.
- Dynamic and kinematic object meshes are automatically decomposed in a set of convex shapes which constitute an approximation of the concave shape.
- The obtained convex hulls are considered a single compound when simulating the dynamic or kinematic objects.
- The convex decomposition code has been included in the XVR plugin used to export AAM files from 3ds Max.
- A new version of the plugin was produced, which is the one to use when working with the new XVR engine.



Physical Simulation

Example: object falling on another object

```
// 6_physics
function OnInit(params) {
      [...]
      PhysicsInit();

      obj1 = XObj("data/Stone1.AAM");
      obj1.SetPosition(0.0,1.5,0.0);
      obj1.RotateGlobal(45.0,1.0,0.0,0.0);
      obj2 = XObj("data/Floor.AAM");
      obj2.RotateGlobal(-90.0,1.0,0.0,0.0);
      obj2.SetPosition(0.0,-1.5,0.0);

      PhysicsPrepare();

      PhysicsStart();
}
```

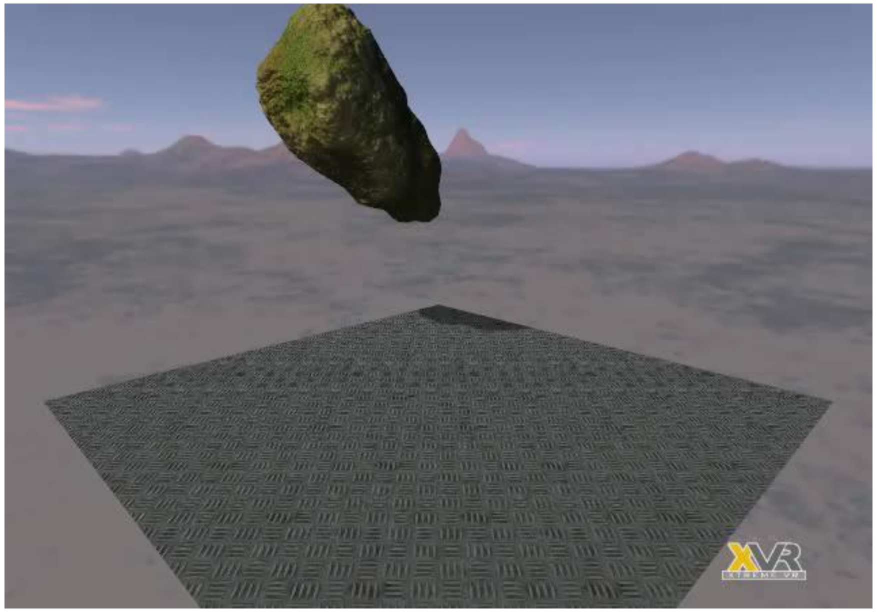


- In order to choose which objects need to be simulated, the user must specify an optional parameter to the PhysicsInit() function.
- The simulation needs to be prepared and then started, but the preparation happens automatically if not done before PhysicsStart() (which will start the simulation).

Physical Simulation

Example: adding a joint

```
// 7_physics
function OnInit(params) {
      [...]
      PhysicsInit();

      obj1 = XObj("data/Stone1.AAM");
      obj1.SetPosition(1.3,1.5,0.0);
      obj1.RotateGlobal(45.0,1.0,0.0,0.0);
      obj2 = XObj("data/Floor.AAM");
      obj2.RotateGlobal(-90.0,1.0,0.0,0.0);
      obj2.SetPosition(0.0,-1.5,0.0);

      PhysicsPrepare();
      joint = PhysicsAddRevoluteJoint(
            obj1, 0.0, 1.5, 0.0, 0.0, 0.0, 1.0);
      PhysicsStart();
}
```

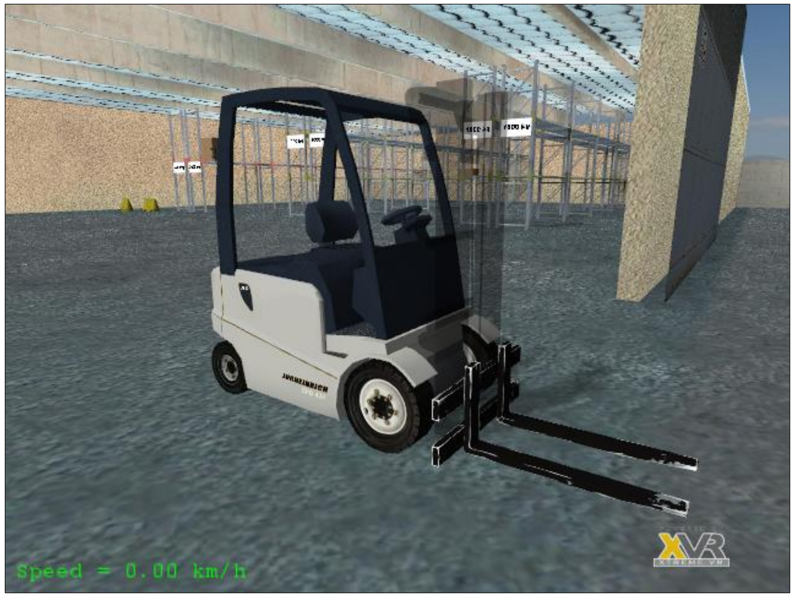


- Joints must be added after the simulation has been prepared.
- Joints may be added and removed at any time, and some support springs, limits and motors.
- Joints may be created between two objects, or between an object and the world (as in our case).

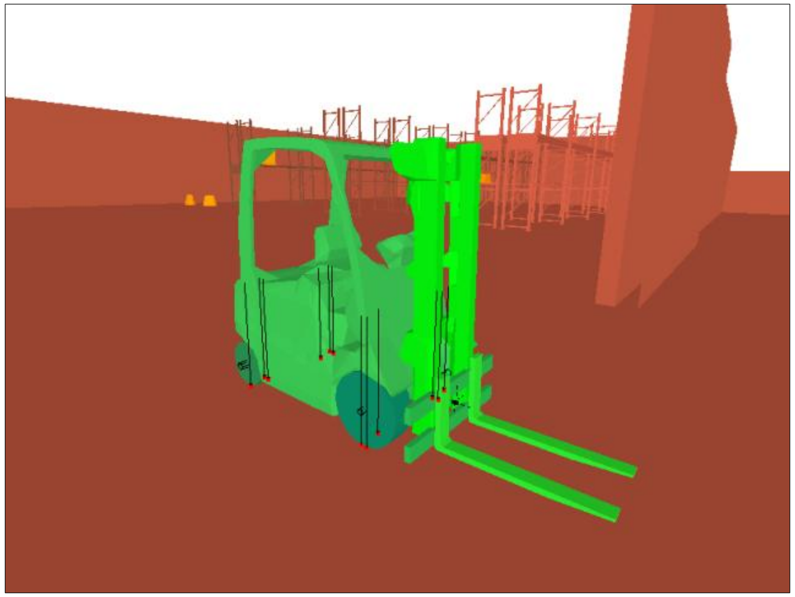
Physical Simulation

Debugging physical simulation

- Because joints are invisible and shapes used for physical simulation are different than the ones used for graphics, debugging physical simulation can be hard.
- The physical equivalent of an XVR scene can be viewed using the Nvidia **PhysX Visual Debugger** with a debug version of XVR (including the debug version of the VR3Lib).



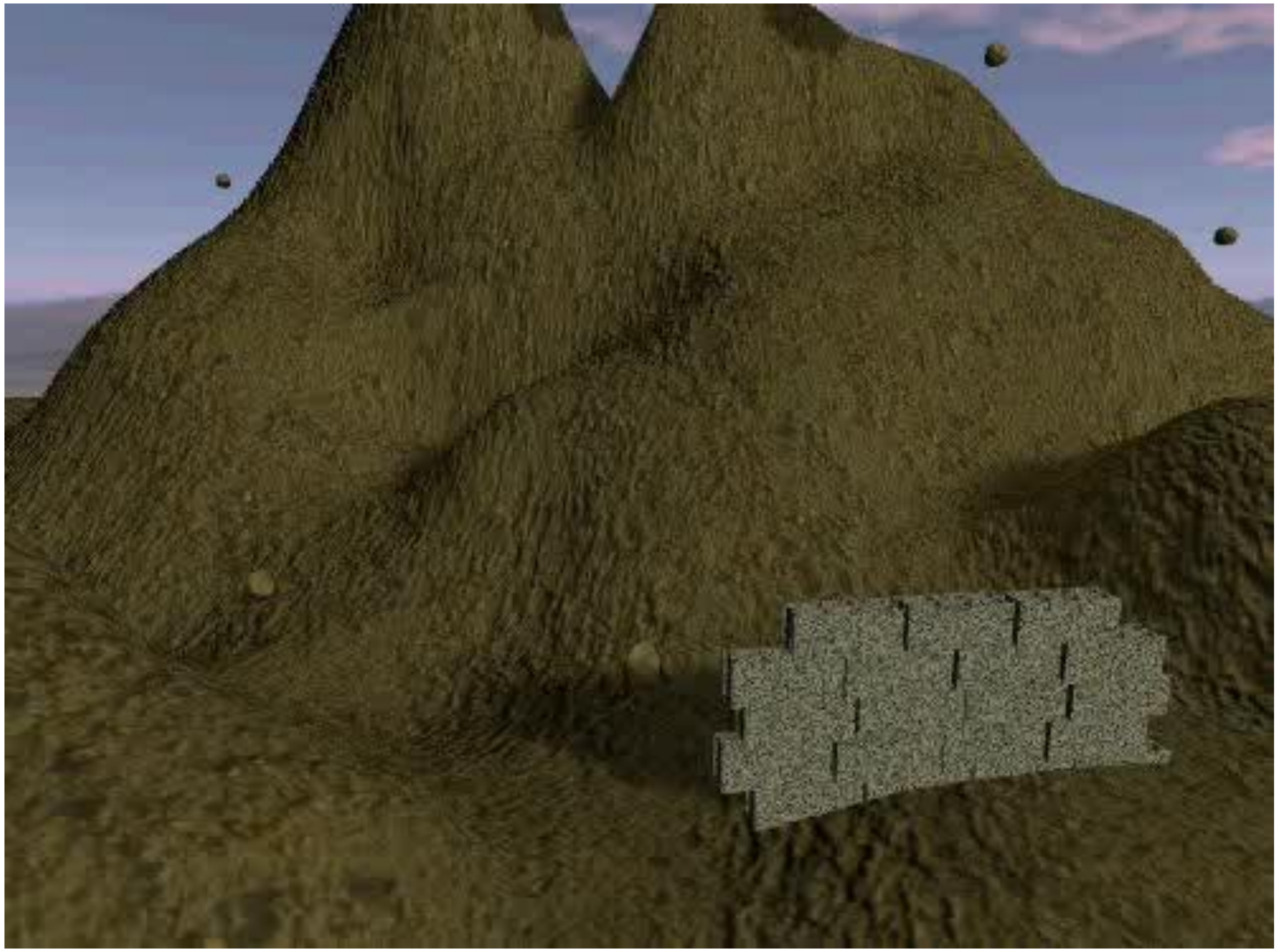
↔



Physical Simulation
Results

Complex physically animated scenes are easily handled with the VR3Lib.

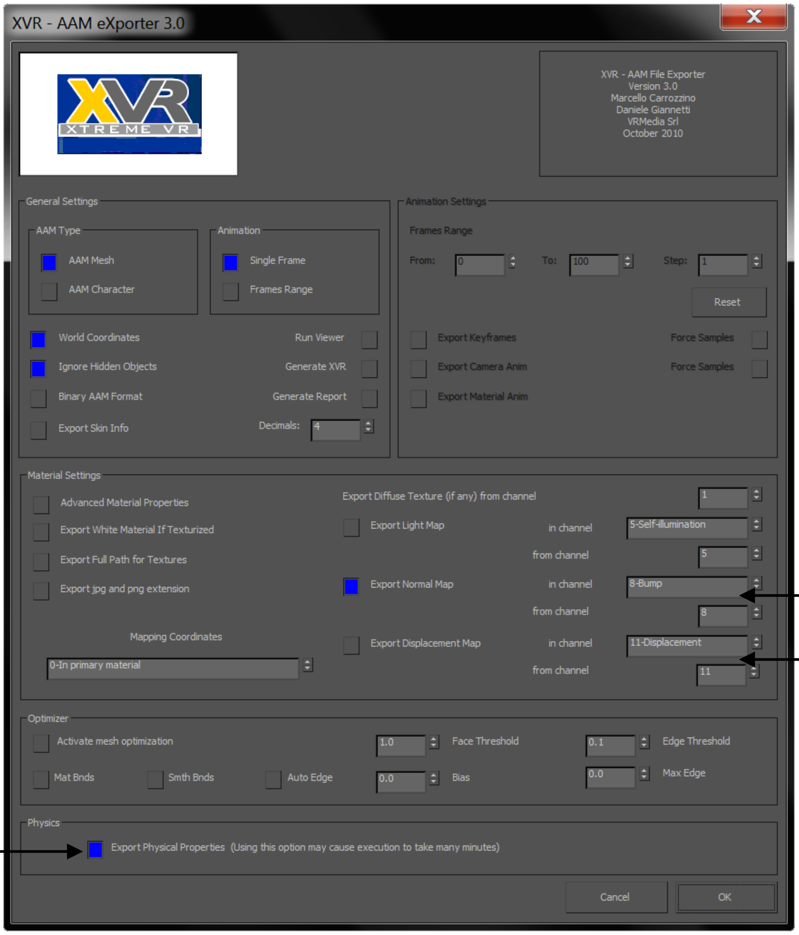
Physical simulation will hardly ever affect frame rate if a multicore processor is available.



Exporting AAM Files

The new 3ds Max exportation plugin must be used to produce AAM files to feed the new engine. Apart from physical properties, the updated plugin is also necessary to access the new texture-based shading techniques.

Flag used to export physical properties (necessary for physical simulation of objects exported).



XVR - AAM eXporter 3.0

XVR - AAM File Exporter
Version 3.0
Marcello Carrozzino
Daniele Giannetti
VRMedia Srl
October 2010

General Settings

AAM Type
- [✓] AAM Mesh
- [ ] AAM Character

Animation
- [✓] Single Frame
- [ ] Frames Range

- [✓] World Coordinates
- [✓] Ignore Hidden Objects
- [ ] Binary AAM Format
- [ ] Export Skin Info
- Run Viewer [ ]
- Generate XVR [ ]
- Generate Report [ ]
- Decimals: 4

Animation Settings

Frames Range
From: 0 To: 100 Step: 1
Reset

- [ ] Export Keyframes — Force Samples [ ]
- [ ] Export Camera Anim — Force Samples [ ]
- [ ] Export Material Anim

Material Settings

- [ ] Advanced Material Properties
- [ ] Export White Material If Texturized
- [ ] Export Full Path for Textures
- [ ] Export jpg and png extension

Mapping Coordinates: 0-In primary material

Export Diffuse Texture (if any) from channel: 1
- [ ] Export Light Map — in channel: 5-Self-illumination — from channel: 5
- [✓] Export Normal Map — in channel: 8-Bump — from channel: 8
- [ ] Export Displacement Map — in channel: 11-Displacement — from channel: 11

Optimizer

- [ ] Activate mesh optimization
- 1.0 Face Threshold
- 0.1 Edge Threshold
- [ ] Mat Bnds
- [ ] Smth Bnds
- [ ] Auto Edge
- 0.0 Bias
- 0.0 Max Edge

Physics

- [✓] Export Physical Properties (Using this option may cause execution to take many minutes)

Cancel OK

Controls to export textures for the new techniques (normal and displacement mapping).

Conclusion and Future Work

- The VR3Lib is not yet fully integrated in the XVR framework (work in progress).
- More capabilities might be added to the new XVR engine before being distributed to the community (e.g. support to physical simulation of non-rigid bodies).
- As the new engine was written using one of the latest OpenGL version (and only core functions), support to the functionalities is guaranteed for the future.
- Some work still to do (e.g. add full support to animation and characters).
- The new XVR version is currently being internally used for some research projects.

Thank you!