Design and Realization of a Software Library for Interactive Visualization of Complex Virtual Environments

Daniele Giannetti

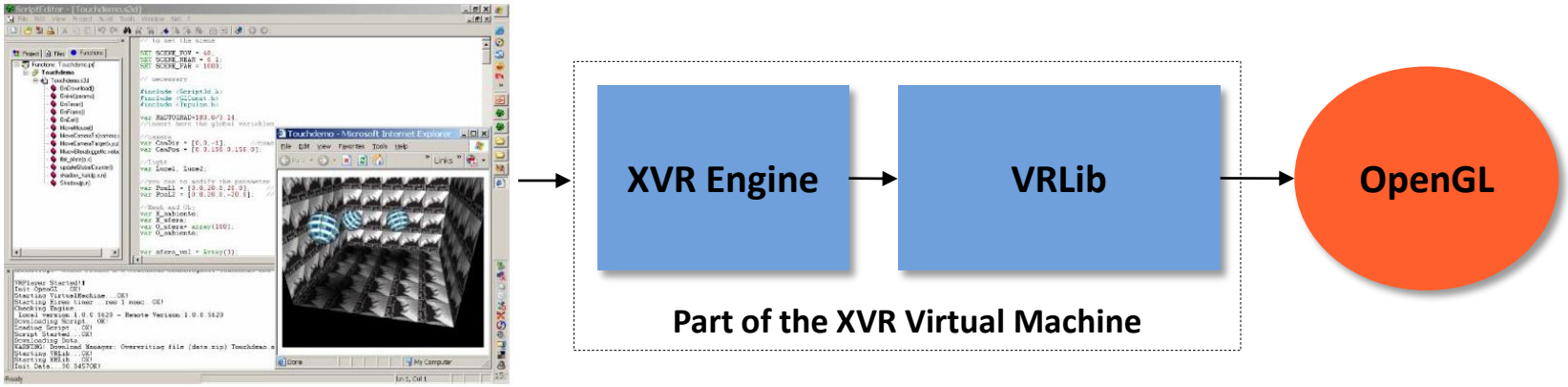
Introduction
Virtual Reality and PERCRO lab

- **Virtual Reality** – nonexistent environments built using complex computerized systems, where the user feels to be part of the virtual world.
- **PERCRO (PERCeptual RObotics) lab** – research laboratory working in the field of virtual reality systems.

Introduction
XVR and VRLib

- **XVR (eXtreme Virtual Reality)** – virtual reality applications development environment built and used at PERCRO.

- **VRLib (Virtual Reality Library)** – real-time rendering library used by XVR to visualize 3D scenes, based on the OpenGL graphics system.



- **OpenGL (Open Graphics Library)** – software interface to graphics hardware used to produce pictures of virtual scenes.

Introduction

VR3Lib: a new VRLib [1]

New graphics programming paradigm based on **shaders**.

OpenGL **Deprecation Model**: old features removed moving towards a fully programmable pipeline.

VRLib requires deep refactoring in order to retain its status of high quality and modern graphics engine.

Need for built-in and easy to use access to modern rendering techniques in order to obtain realistic virtual environments.

Need for built-in physical simulation in order to easily build physically realistic virtual reality application.

Development of a new low level real-time rendering library: the **VR3Lib**.

Introduction
VR3Lib: a new VRLib [2]

- The VR3Lib is the new version of the VRLib library, result of a complete rewriting of the previous engine, including built-in support for:
  - Physical simulation of virtual objects (rigid bodies)
  - Many shader-based modern rendering techniques

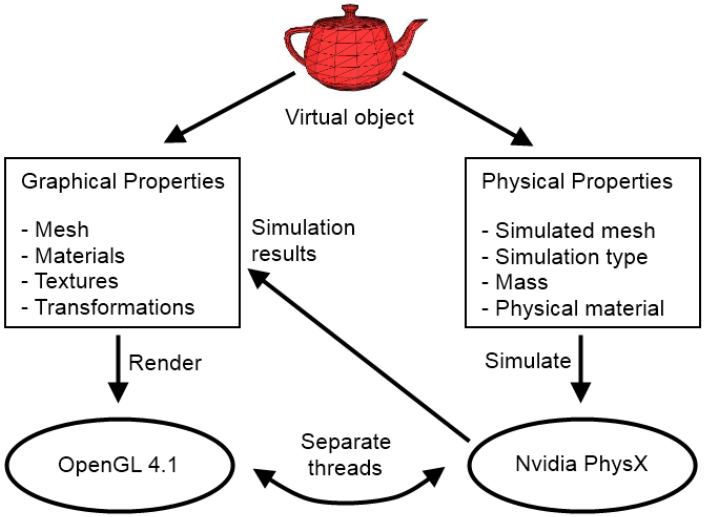
  Those two aspects represent the true innovation obtained using the VR3Lib.
- The new library was built maintaining and extending the API from the previous version (currently used by the XVR technology and applied in several research labs across EU) in order to allow easy integration of the VR3Lib as a VRLib replacement.
- Because XVR is often applied in interactive web applications, the new VR3Lib is (as the previous version) lightweight, only including needed functionalities.

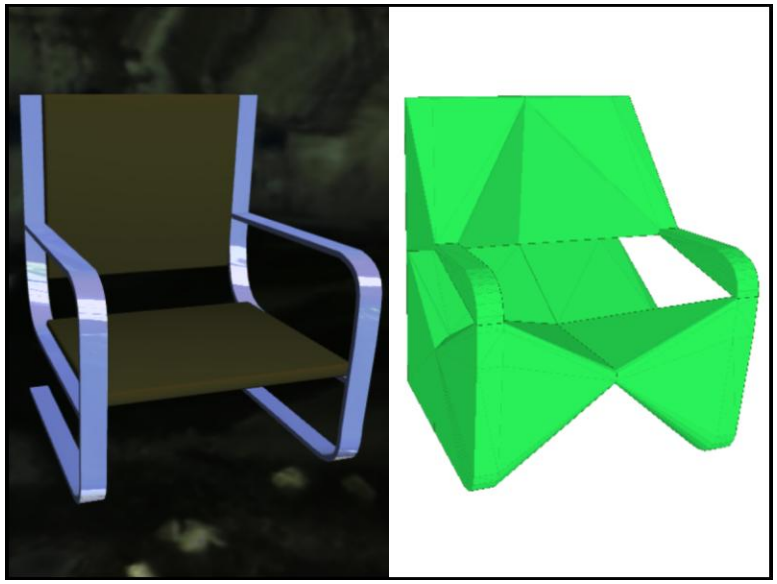
Joining Graphics and Physics
The Nvidia PhysX Engine

- We decided to use a free, fast and reliable solution to obtain physical simulation of virtual objects: the Nvidia **PhysX** engine.
- Physical simulation is carried on by PhysX while the main rendering cycle proceeds using VR3Lib draw calls (multithreaded and efficient solution).
- The physical simulation thread is managed by PhysX directly and the VR3Lib only needs to synchronize with it when simulation results are needed to update scene properties (such as object position and rotation).

The PhysX engine is currently available only for Windows operating systems. VR3Lib physical simulation services will not be available on different operating systems.

Joining Graphics and Physics
Two Representations for Virtual Objects





- Two descriptions for virtual objects:
  - Graphical description (mesh, materials, textures, …)
  - Physical description (physical mesh, mass, …)
- Potentially **different**: complex objects may have a very simple shape for physical simulation.

Joining Graphics and Physics
Extending the AAM File Format

The **AAM** file format is the format used by the VRLib library family to obtain data on virtual objects (shape, material, etc…).

The file format was extended to include a physical description of virtual objects.

```
AAM_MESH
MATERIALS
MatCount: 1
Mat# 0
{
    Name: 03 - Default
    Class: Standard
    Am: 0.8196 0.7490 0.5922
    Di: 0.8196 0.7490 0.5922
    Sp: 0.0000 0.0000 0.0000
    Tr: 0.0000
    Sh: 0.1000
...
```

- The physical description and graphical description of the virtual objects are tightly coupled together.
- The physical representation for a virtual object is automatically generated depending on the type of simulation desired for the particular object.
- AAM files are created with a modeling software such as 3ds Max, using a GUI-provided exportation plug-in.

Obtaining Visual Realism
Direct Illumination of Virtual Objects

Considering point light sources, compute realistic lighting of virtual objects.
We use the well-known **Phong reflection model** to obtain the resulting color.

Flat surfaces are shaded simply using the normals of each polygon.

(**flat shading**)

Curved surfaces are shaded considering an interpolated normal for each fragment.

(**Phong interpolation**)

Obtaining Visual Realism
Image-Based Lighting [1]

Image-Based Lighting (IBL): Obtaining realistic illumination using real world pictures instead of point light sources.

IBL is a family of techniques, some of the approaches commonly used are only valid for non-interactive applications.

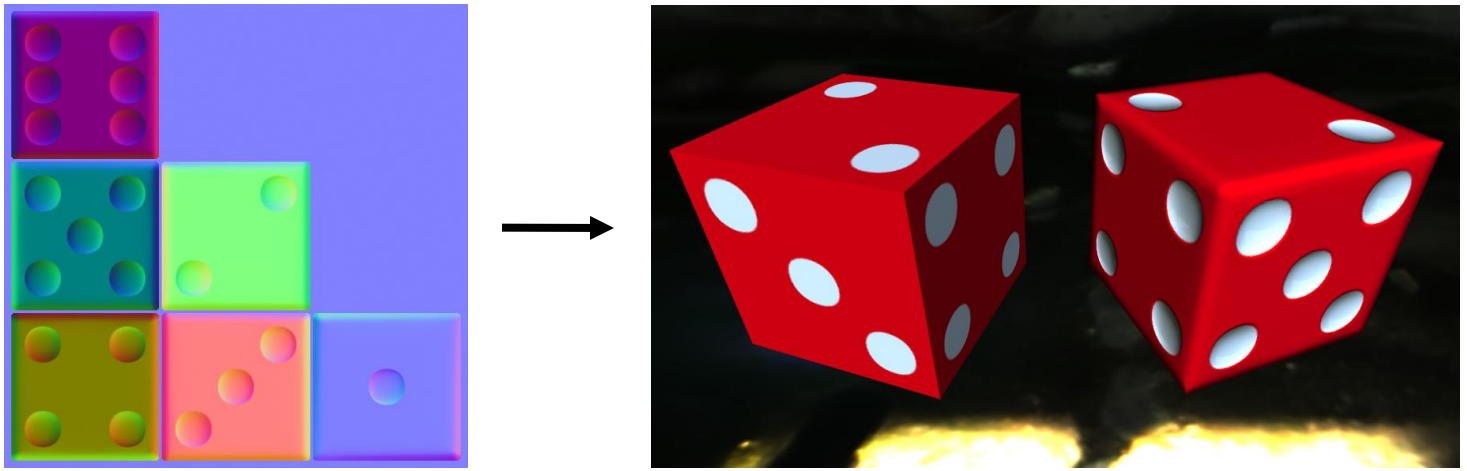
Usually, **High Dynamic Range (HDR)** digital pictures are used to compute illumination of virtual scenes. Those pictures are obtained using advanced photographic techniques.

Obtaining Visual Realism
Image-Based Lighting [2]

- We use a **cube environment mapping** approach to image-based lighting: real world pictures are transformed in cube environment maps to be used for shading, and a set of cube maps is used to compute illumination of objects.

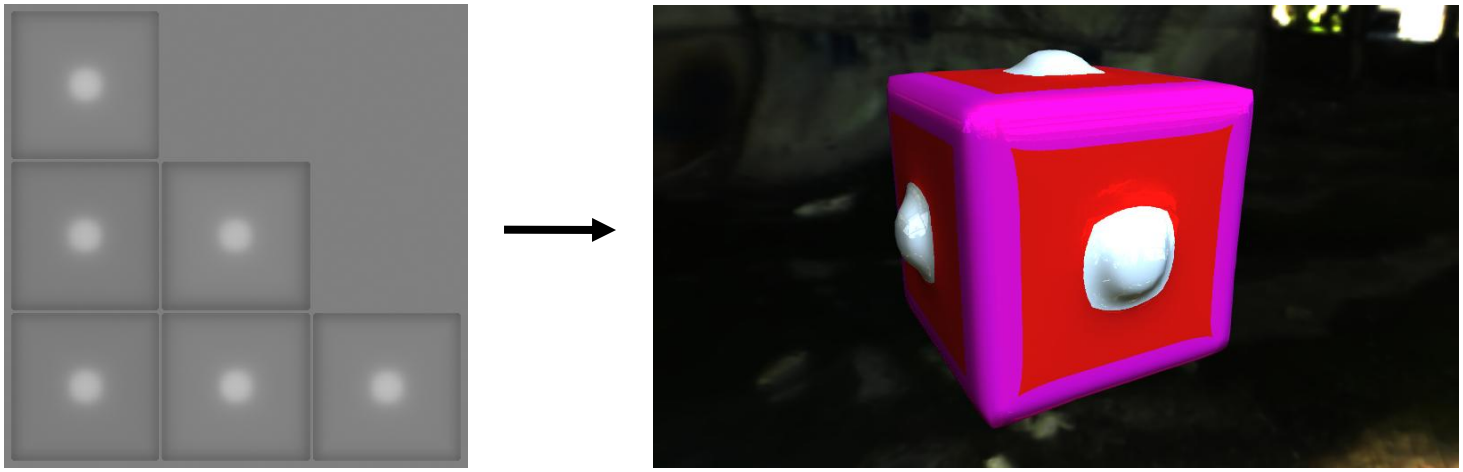
- We usually apply two cube environment maps:
  - Diffuse cube environment map
  - Specular cube environment map

Obtaining Visual Realism
Surface Details using Normal Maps



- Highly detailed objects (with a lot of polygons) may introduce a heavy load on the rendering pipeline, leading to a reduced frame rate.
- The normal mapping technique forges surface details using **image-driven normal perturbation**.
- Significantly improve visual realism with a very small additional computational effort.

Obtaining Visual Realism
Surface Details using Displacement Maps



- With displacement mapping, geometry is dynamically altered **at rendering time**: we use the highly parallel floating point architecture of modern graphics cards to efficiently generate the surface details tessellating and displacing the geometry.
- Usually combined with the normal mapping technique to forge micro-details that do not require displacement mapping.
- Computationally expensive.

Obtaining Visual Realism
Casting Shadows in Real-Time

Two well-known basic real-time shadow casting algorithms.

Shadow Mapping

Render the scene from the point of view of the light source, a shadow map is obtained to be used during final scene rendering.

Shadow Volumes

Silhouette edges are extended to obtain shadow volume polygons that confine shadowed regions of the 3D space.

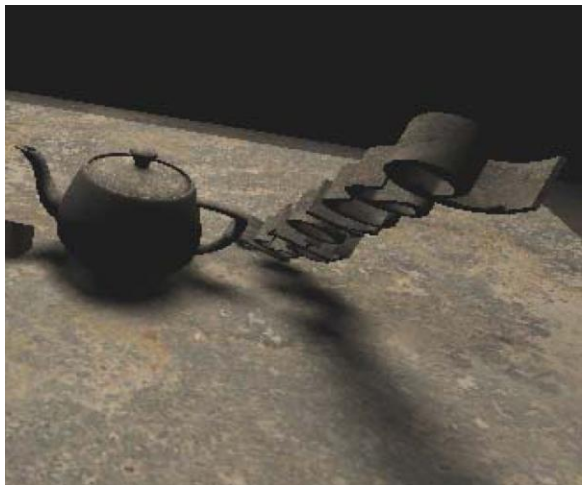
Hard-edged shadows are obtained, but to achieve realistic results we need to produce soft-edged shadows with penumbra regions (or **soft shadows**).

Obtaining Visual Realism
Soft Shadow Mapping [1]

Many different techniques to obtain soft shadows using shadow mapping.

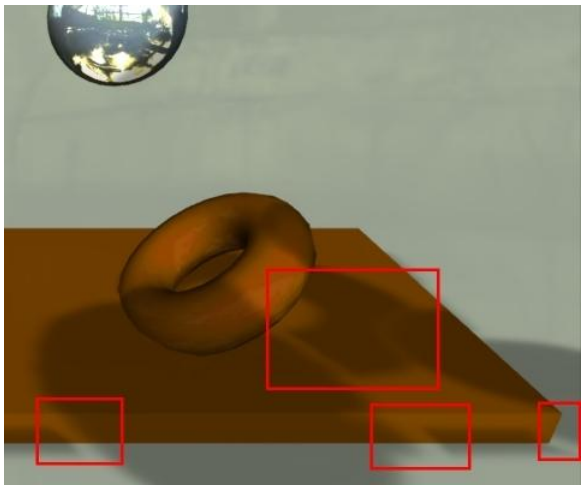
PCSS (2005)

Filter the shadow map at rendering time using PCF with variable filter size.



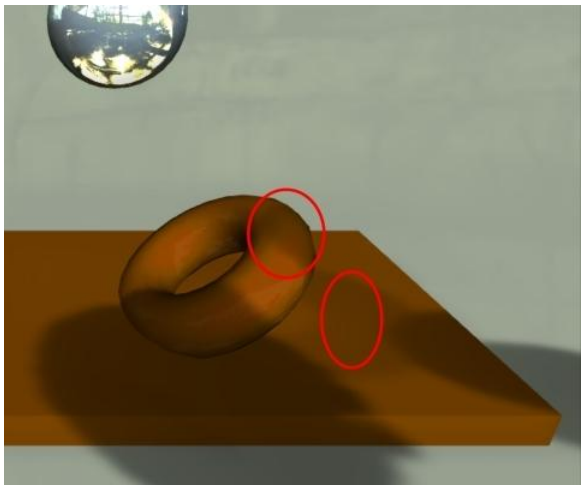
VSM (2006)

Pre-filter the shadow map considering the result as a set of probability distributions of depth.



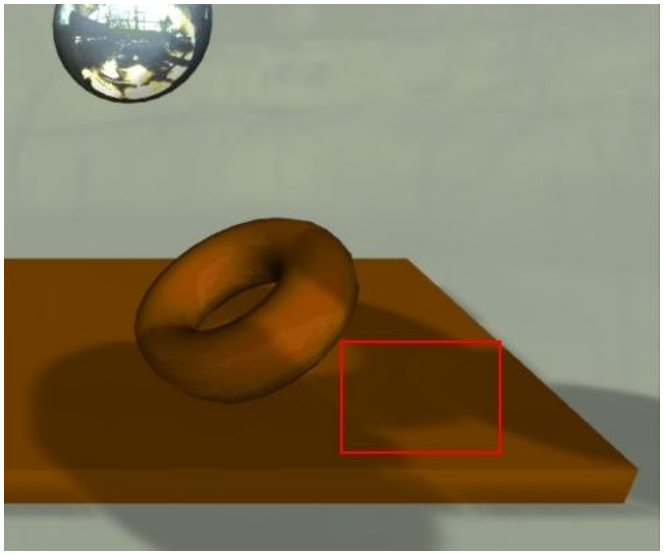
ESM (2008)

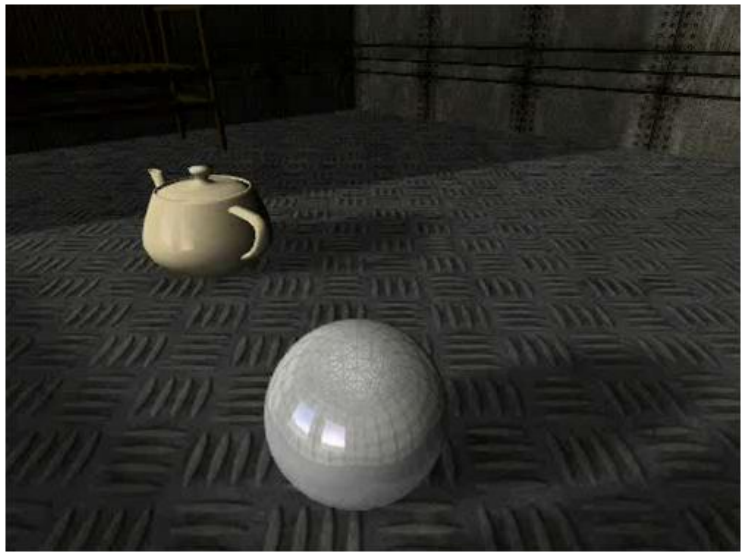
Same as VSM, but only a single channel is needed and filtering results are used differently.



Obtaining Visual Realism
Soft Shadow Mapping [2]

- An innovative technique: **EVSM (Exponential Variance Shadow Mapping)**.
- EVSM improves upon VSM and ESM by reducing artifacts obtained:
  - Light bleeding
  - Artifacts for non-planar receivers
- Introduced in the VR3Lib as a built-in functionality for soft shadow casting.





Obtaining Visual Realism
Implementation of the Previous Techniques

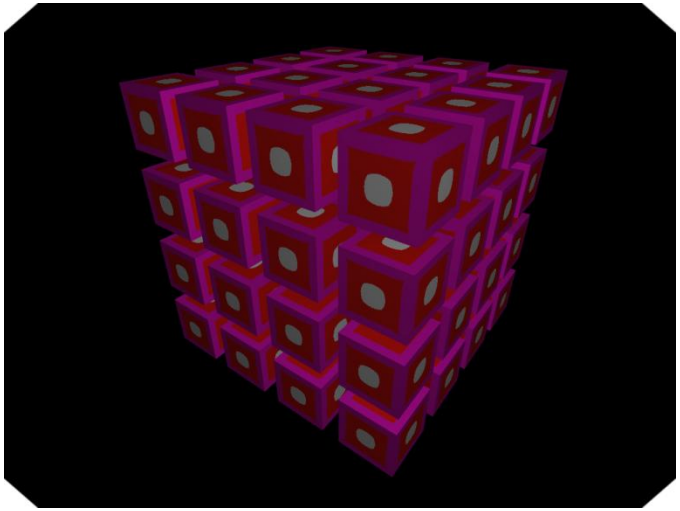
- All the presented rendering algorithms (and more) have been introduced in the VR3Lib as built-in and easy to use features.
- Most of the work in realizing the presented techniques was writing the shader programs executed on the GPU during rendering.
- **GLSL (OpenGL Shading Language)** is the shading language of choice when working with OpenGL (direct support for compilation, linking and loading).
- The VR3Lib core is a software module capable of managing and dynamically loading the shader programs needed for rendering.
- The AAM file format (and therefore the 3ds Max exporter) was also extended to include data needed for the new rendering techniques (such as the name of the normal map file and the coordinates for its application).

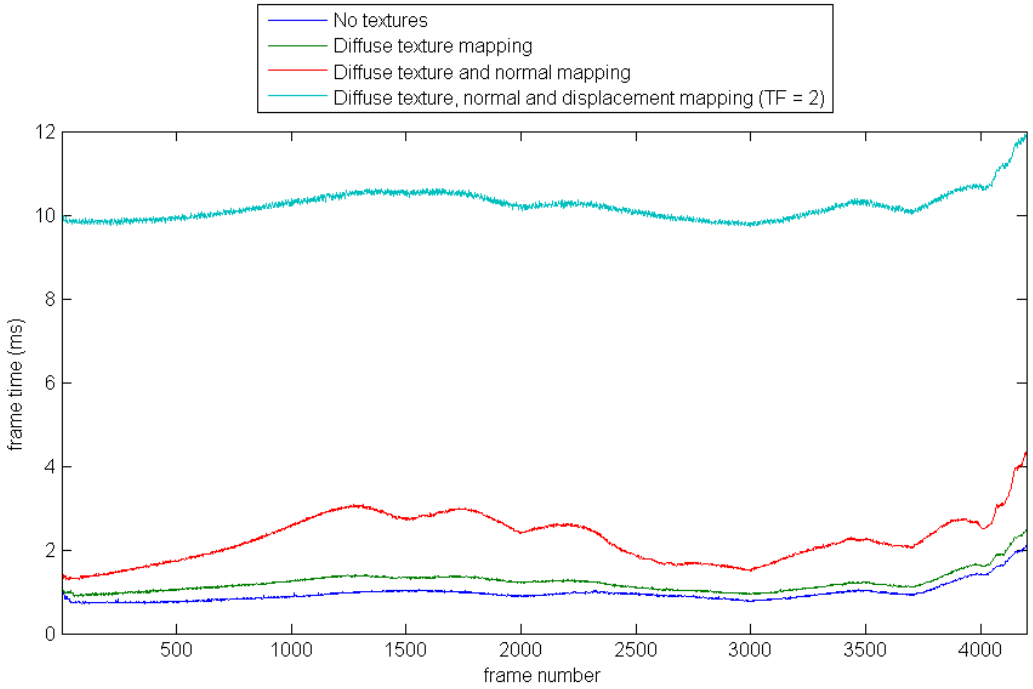
Testing and Performance
Geometric Complexity

- Interactive virtual reality applications performance is always measured in terms of time needed to draw a single frame (**frame time**) or its reciprocal (**frame rate**).
- VR3Lib – based applications performance also depends upon the solution used to manage the OpenGL window and the framebuffer.
- VR3Lib performances depending on the number of polygons in the scene:
  - 10k triangles → frame time < 2 ms
  - 100k triangles → frame time < 4 ms
  - 650k triangles → frame time < 8 ms
  - 3M triangles → frame time ≈ 31 ms

  The effective values also depend upon the on-screen size of the visualized virtual objects.

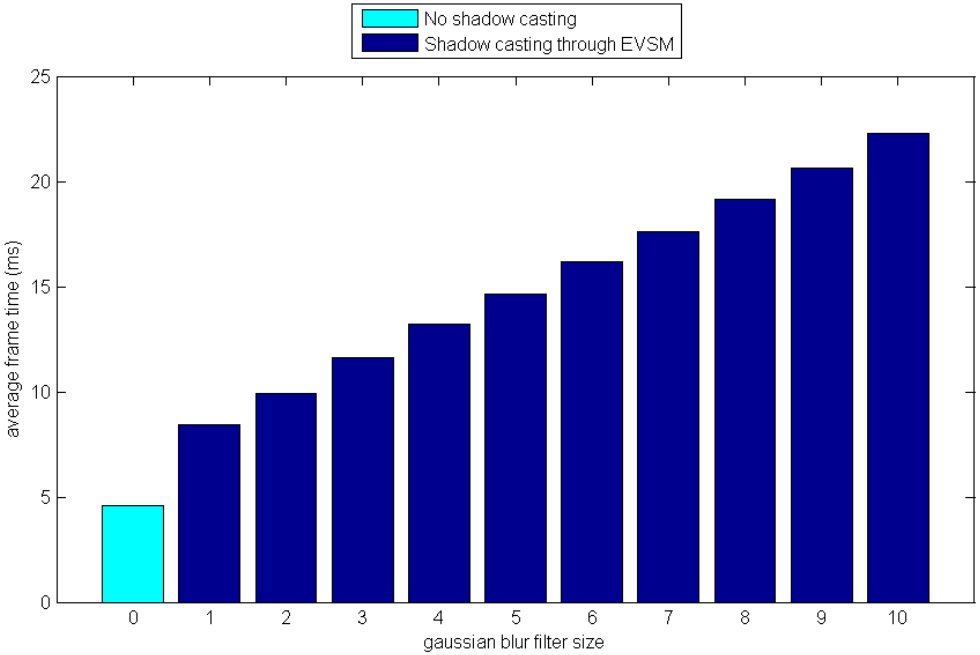
Testing and Performance
Shading Techniques





- 512x512 textures, 1024x768 viewport, 6912 triangles, environment mapping.
- Simple geometry to highlight differences.
- Displacement mapping performance is rather disappointing, we plan to investigate on adaptive geometry tessellation using different approaches (such as tessellation shaders).

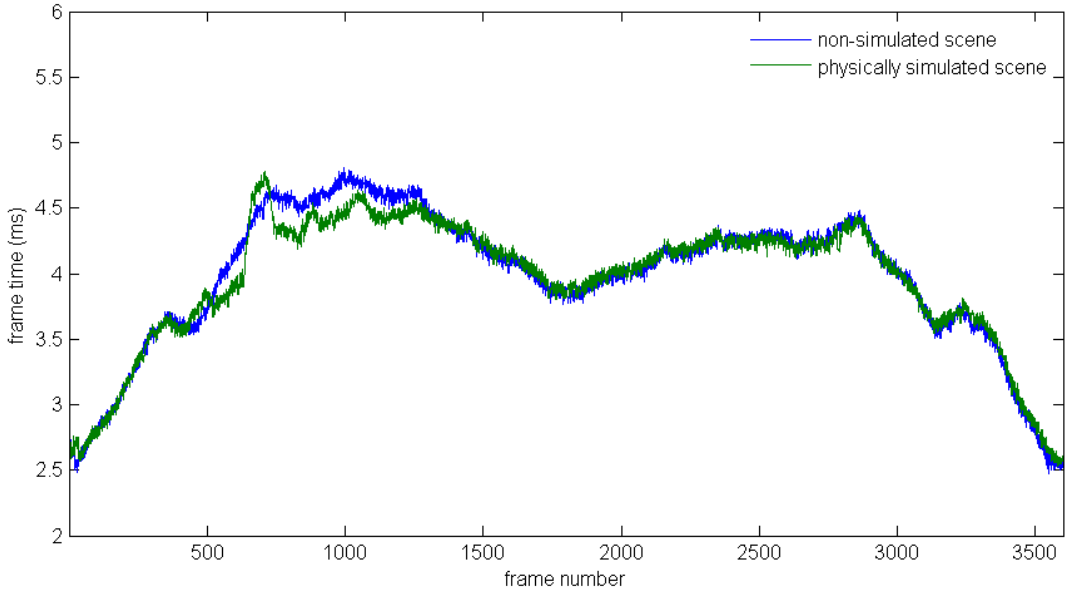
Testing and Performance
Shadowing



- 512x512 shadow map, 1024x768 viewport, 10572 triangles.
- EVSM applied with gaussian blur pre-filtering of the shadow map, average frame time grows linearly with filter size (separable filter).
- Our technique is efficient and produces good looking soft shadows, more expensive methods may not be fast enough for interactive applications.

Testing and Performance
Physics Simulation

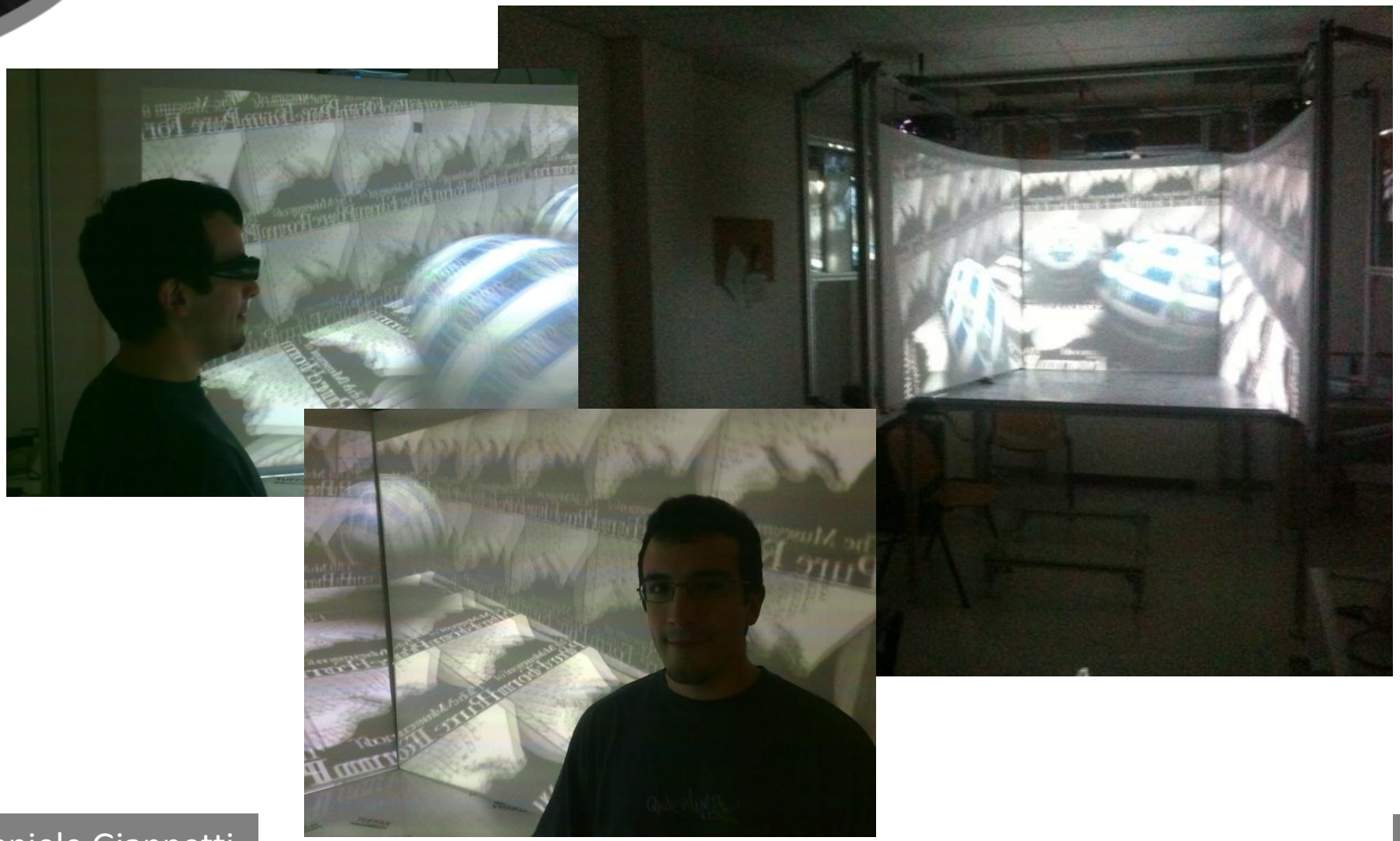
- Simulation takes place in a separate (PhysX managed) thread.
- If results are not ready when needed, we skip the objects state update phase and proceed to the scene rendering.
- The physical simulation of the scene (even when extremely complex) will never affect frame time if the two threads may execute at the same time (but the simulation may become fairly inaccurate).



Conclusion and Future Work

- The realized library is a full-featured graphics and physics standalone engine to use when building virtual reality applications.
- The next planned step is integration of the VR3Lib library into the XVR virtual reality application development system.
- The VR3Lib will need continuous maintenance in order to retain its status of modern and last-generation virtual reality engine.
- Some work may still be done in order to improve the rendering capabilities included in the new library (and introduce more functionalities). As in the previous version, VR3Lib functionalities may also be extended by means of user-provided shaders.
- We may add support for physical simulation of non-rigid bodies (clothes, fluids and soft bodies).
- Ease of use is one of the key requirements that will lead future VR3Lib development.

Example of a CAVE system




Thank you!