Shadow Maps and Projective Texturing In X3D

Michalis Kamburelis
michalis.kambi@gmail.com

Institute of Computer Science
University of Wrocław, Poland

Web3D 2010

Uniwersytet Wrocławski

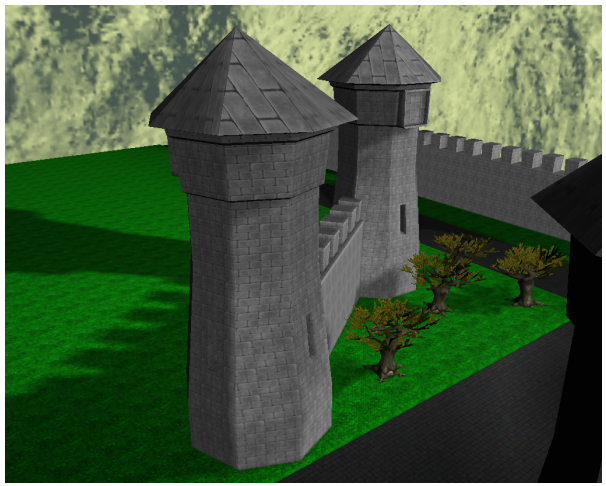
Outline

Uniwersytet Wrocławski

Outline

Uniwersytet Wrocławski

We Want The Scenes To Look Pretty

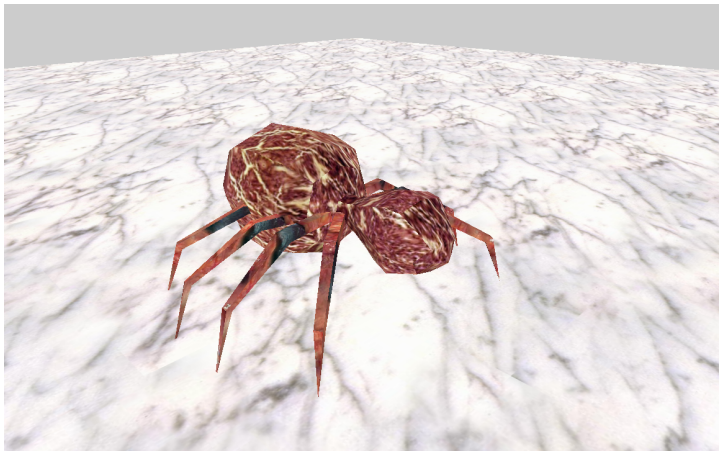


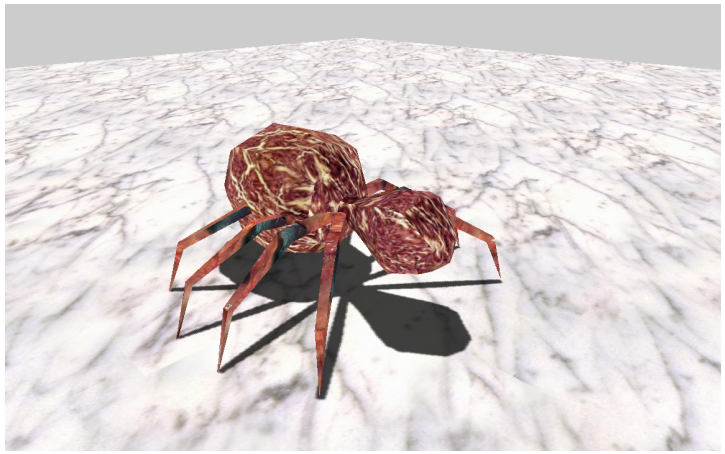
- Shadows make the scene look realistic.
- Shadows automatically make the impression that „more stuff is happening” on the screen. This is even more true for dynamic scenes, where shadows also change dynamically.

Uniwersytet Wrocławski

We Want The User To Understand What Is Displayed

Shadows help the mind to determine the positions of the 3D objects.
Is this spider hovering or standing on the ground?





Uniwersytet Wrocławski

Previous Work

Both Bitmanagement and Octaga proposed their own shadows extensions. We argue that our extensions are a little better :)

- Easier for non-technical authors
- Flexible for advanced authors
- Nice for implementors

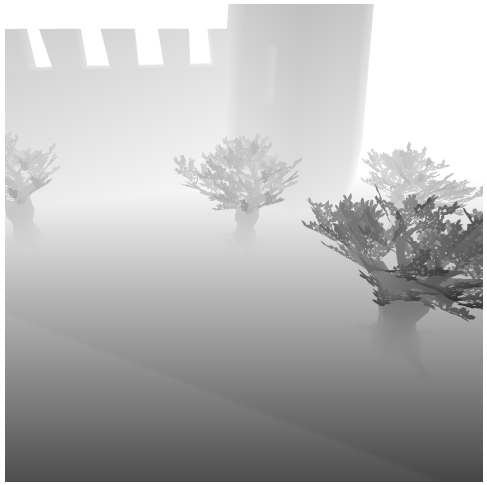
Of course, we would like to see the shadows extensions included in the X3D specification.

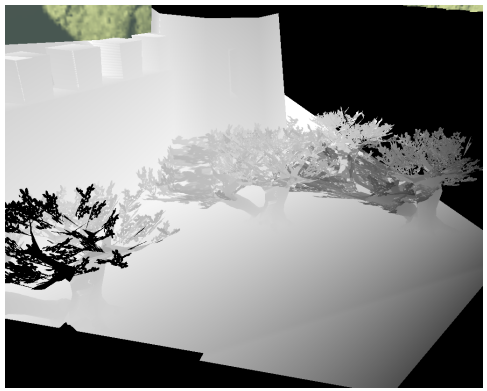
Uniwersytet Wrocławski

Outline

Uniwersytet Wrocławski

Shadow Maps Algorithm Overview

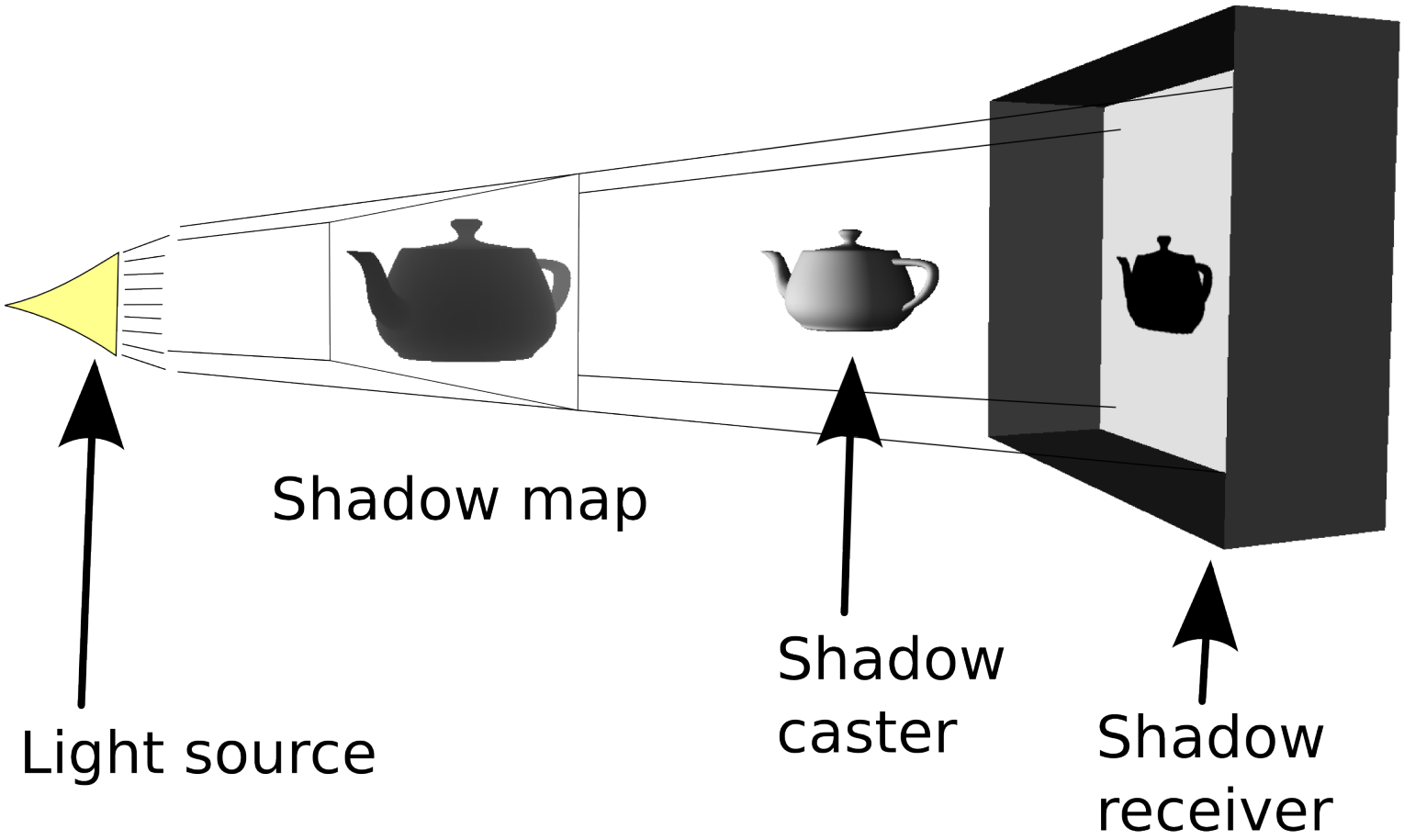




1. Render a shadow map from a light source.
2. Project a shadow map when rendering the actual view for the screen. This means using a shadow map texture, and calculating a texture coordinates such that you know the distance from the light at each screen pixel.
3. Compare depth value from the shadow map with the actual depth.
   Object in the shadow must be behind an object obscuring the light.

Uniwersytet Wrocławski

Shadow Maps Terms



Uniwersytet Wrocławski

Outline

Uniwersytet Wrocławski

Define Shadow Receivers

New field for the Appearance node

```
MFNode  []  receiveShadows  []
  # [X3DLightNode] list
```

- Very easy to use.
  Ideally, the only thing really needed by authors for simplest purposes.
- So simple, it's suitable for any shadow algorithm.
- Light contribution (whole, including ambient) is scaled to zero where the light is in the shadow. This way object in shadow looks the same as object outside of light's radius, so things look OK.

Uniwersytet Wrocławski

Reasoning: Where To Place Shadow Information?

- **Separate node**: not natural. We want to make shadows commonly used in the scenes, so activating shadows must be easy. Also, separate node is not really needed for flexibility.
- **Casters**: there's not much you want to configure on shadow casters. You just want to „treat everything as shadow caster by default".
- **Lights**: yes, put some information there, but not flexible enough.
- **Shadow receivers** seem like the best choice:
  - Natural for authors: shadows change the look of the shadow receiver.
  - Natural for implementors: shadow receivers must be rendered in a special way.

Uniwersytet Wrocławski

Shadow Casters

By default, everything casts a shadow. Simple global configuration: you can disable casting any shadows from this shape.

New field for the Appearance node

`SFBool [in,out] shadowCaster TRUE`

What if this is too simple? For complicated cases, another field like `doNotCastShadows` can be added, to disable casting shadows for some chosen lights and/or some chosen receivers. In practice, this wasn't needed on our scenes.

Uniwersytet Wrocławski

Shadow Map Parameters

In principle, every light by default is capable of casting shadows. So you don't *have* to do anything. However, you can tweak some shadow maps stuff:

New field for the X3DLightNode

```
SFNode   []   defaultShadowMap   NULL
  # [GeneratedShadowMap]
```

The way this works will be clear later, when we describe `GeneratedShadowMap` node. For now, we just note that this allows you to:

- manually control when the shadow map is updated,
- change shadow map size,
- indicate that you want to visualize shadow maps.

Uniwersytet Wrocławski

Lights Projection Parameters

For shadow maps, we have to be able to treat lights as cameras. So we need some projection parameters. When you use `receiveShadows`, the browser should be able to calculate suitable values for fields below automatically.
But you can tweak them, for complicated cases and low-level usage.

New fields for X3DLightNode

```
SFFloat   [in,out]   projectionNear 0
SFFloat   [in,out]   projectionFar  0
SFVec3f   [in,out]   up 0 0 0
```

...and some additional fields for the `DirectionalLight` and `SpotLight`, see the paper for details.

Uniwersytet Wrocławski

Outline

Uniwersytet Wrocławski

Why Low-level Extensions?

A couple of cooperating extensions to achieve shadow maps.

- Implementors:
  - Very easy to handle in the renderer.
  - Open a way to handle high-level extensions by transforming to low-level.
- Authors:
  - Not easy for authors: you need to understand how shadow maps work.
  - But: you can use own shaders (GLSL etc.)
  - But: you can use projective texturing for other purposes.

Uniwersytet Wrocławski

Example of Low-level Extensions Usage

```
DEF MySpot SpotLight {
  location 0 0 10    direction 0 0 -1
  projectionNear 1    projectionFar 20   }
Shape {
  appearance Appearance {
    texture GeneratedShadowMap {
      light USE MySpot    update "ALWAYS" } }
  geometry IndexedFaceSet {
    texCoord ProjectedTextureCoordinate {
      projector USE MySpot }
    # ... other IndexedFaceSet fields
  } }
```

Uniwersytet Wrocławski

How Is This Related To High-level Extensions?

The low-level extensions cooperate very nicely with high-level extensions described previously:

- `receiveShadows` and `defaultShadowMap` may be implemented by transforming the scene to use low-level extensions.
  How to transform: For each shape with `receiveShadows`, add to it's textures a `GeneratedShadowMap` node and add to it's tex coords a `ProjectedTextureCoordinate`. Details in the paper.
- Rest (light's projection, shadow casters) is simply used as-is.

Uniwersytet Wrocławski

New X3D Node: Generated Shadow Map

You can place this inside e.g. a `texture` field, or as a texture inside shader node (like `ComposedShader` for GLSL):

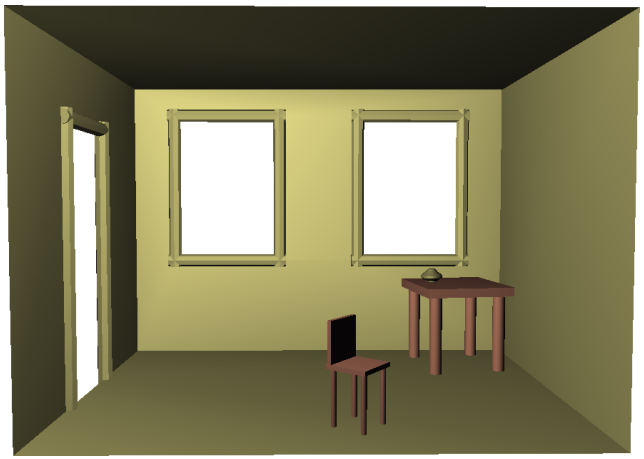
GeneratedShadowMap : X3DTextureNode

```
SFString   [in,out]  update "NONE"
  # ["NONE" | "ALWAYS" | "NEXT_FRAME_ONLY"]
SFInt32    []        size   128
SFNode     []        light  NULL # [X3DLightNode]
SFFloat    [in,out]  scale  1.1
SFFloat    [in,out]  bias   4.0
SFString   []        compareMode
  "COMPARE_R_LEQUAL" # ["COMPARE_R_LEQUAL"
  # | "COMPARE_R_GEQUAL" | "NONE"]
```

wersytet
ocławski

Projecting Textures Explained

Untextured scene



+

Texture to project

=

Final scene
with projected texture

Final scene,
as seen from
the projector:

Uniwersytet Wrocławski

New X3D Node: Projected Texture Coordinate

You can place this inside e.g. `texCoord` fields:

ProjectedTextureCoordinate : X3DTextureCoordinateNode

```
SFNode  [in,out]  projector  NULL
  # [SpotLight, DirectionalLight,
  # X3DViewpointNode]
```

Generates texture coordinates that project (cast) a texture.

- For shadow maps: project your shadow map from a light source.
- You can also project a normal texture (e.g. color `MovieTexture`).
- You can also project from a viewpoint (e.g. to cast a headlight texture).

Uniwersytet Wrocławski

Outline

Uniwersytet Wrocławski

Summary

High-level extensions to easily get shadows

- Define shadow receivers by the `receiveShadows` field.
- Easy to use, easy to implement basing on low-level extensions.

Low-level extensions to fully control shadow maps

- Define shadow receivers by explicitly adding `GeneratedShadowMap` and `ProjectedTextureCoordinate` to textures and tex coords.
- Easy to implement.
- Projective texturing available also independently.
- Usable with custom shaders.

wersytet
ocławski

Thank you for your attention!

Questions?

Visit our VRML engine on
http://vrmlengine.sourceforge.net/

Uniwersytet Wrocławski