# Real-time Rendering of Parametric Skin Model

Stanislav Fecko

Supervisor: Martin Madaras

April 28th, 2013

# Motivation

Image from "Pirates of the Caribbean: Dead Man's Chest" movie.

# Contents

The **goal** of this work is to create a rendering system capable of realistic **real-time** rendering of various skin materials.

[1] http://www.imdb.com/title/tt0383574
[2] http://luima.com

The **goal** of this work is to create a rendering system capable of realistic **real-time** rendering of various skin materials.

Eugene d'Eon [1] presented a technique of efficient rendering of human skin.
We aim to extend the methods proposed in existing papers to render also different **non-human** skins, as aliens, undead or even more bizzare creatures.

---

[1] http://www.imdb.com/title/tt0383574
[2] http://luima.com

The **goal** of this work is to create a rendering system capable of realistic **real-time** rendering of various skin materials.

Eugene d'Eon [1] presented a technique of efficient rendering of human skin.
We aim to extend the methods proposed in existing papers to render also different **non-human** skins, as aliens, undead or even more bizzare creatures.

For our testing model we have chosen the head of Davy Jones, a character from Pirates of the Caribbean[1] movie.

The model (courtesy of Luis Manuel Morillo[2]) suits our needs because of its rather specific skin: **wet, slimy and oily**.

---

[1] http://www.imdb.com/title/tt0383574
[2] http://luima.com

# Deferred shading

# Deferred shading

**Deferred shading** is a concept of rendering that separates rendering of geometry and shading into two separate passes.

# Deferred shading

**Deferred shading** is a concept of rendering that separates rendering of geometry and shading into two separate passes.

Main advantages of the approach:

- Less shading in complex scenes with high overdraw

# Deferred shading

**Deferred shading** is a concept of rendering that separates rendering of geometry and shading into two separate passes.

Main advantages of the approach:

- Less shading in complex scenes with high overdraw
- Efficient rendering of many small, local lights

# Deferred shading

**Deferred shading** is a concept of rendering that separates rendering of geometry and shading into two separate passes.

Main advantages of the approach:

- Less shading in complex scenes with high overdraw
- Efficient rendering of many small, local lights
- Makes HDR rendering possible

# Deferred shading

**Deferred shading** is a concept of rendering that separates rendering of geometry and shading into two separate passes.

Main advantages of the approach:

- Less shading in complex scenes with high overdraw
- Efficient rendering of many small, local lights
- Makes HDR rendering possible
- Directly supports various screen-space post processing effects

# Deferred shading, FBO

Our implementation uses **framebuffer objects**[3] (FBO).

FBO is an OpenGL structure consisting of a set of (almost) arbitrary **textures**.

We can render into **each** of the textures instead of the backbuffer.
And we can read from **each** of the textures.

[3] http://www.opengl.org/wiki/Framebuffer_Object

# Deferred shading - Killzone Example

In **geometry pass** vertex attributes are rendered into textures:

# Deferred shading - Killzone Example

In **geometry pass** vertex attributes are rendered into textures:

# Deferred shading - Killzone Example

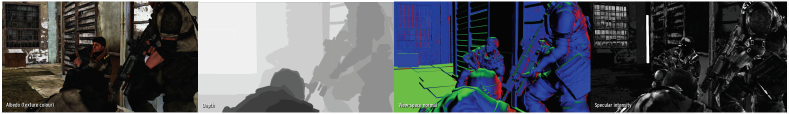
In **geometry pass** vertex attributes are rendered into textures:

# Deferred shading - Killzone Example

In **geometry pass** vertex attributes are rendered into textures:

# Deferred shading - Killzone Example

In **geometry pass** vertex attributes are rendered into textures:



# Deferred shading - Killzone Example

In **shading pass** data is being processed (shading, effects):


Top row: vertex attributes. Bottom: final render. Images from [5].

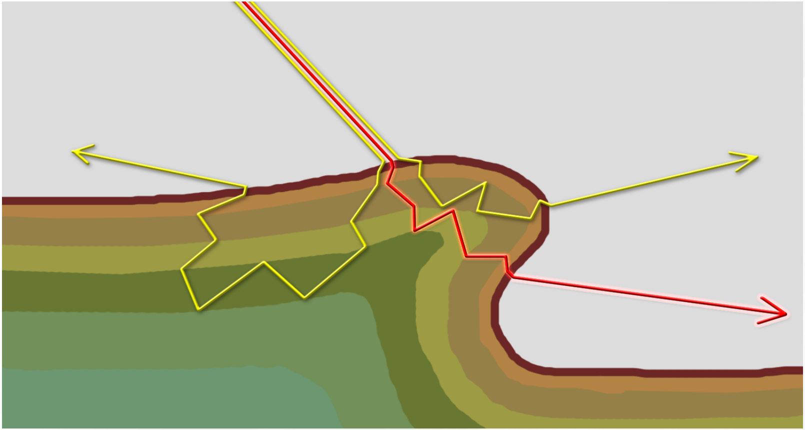
# Translucent materials

# Sub-surface scattering

**Sub-surface scattering** (SSS) is a natural phenomenon observed on translucent materials.

Light entering the volume of the object interacts with the material, **scatters**, and exits the volume at a **different** point and with a different direction.

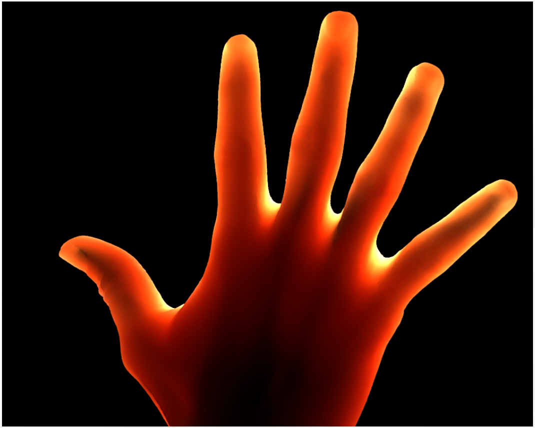
# Sub-surface scattering - Concept



# Sub-surface scattering - Concept

Due to the SSS effect even areas that are not directly lit receive some amount of lighting.

# Sub-surface scattering - Example



High quality translucence. Image from [2].

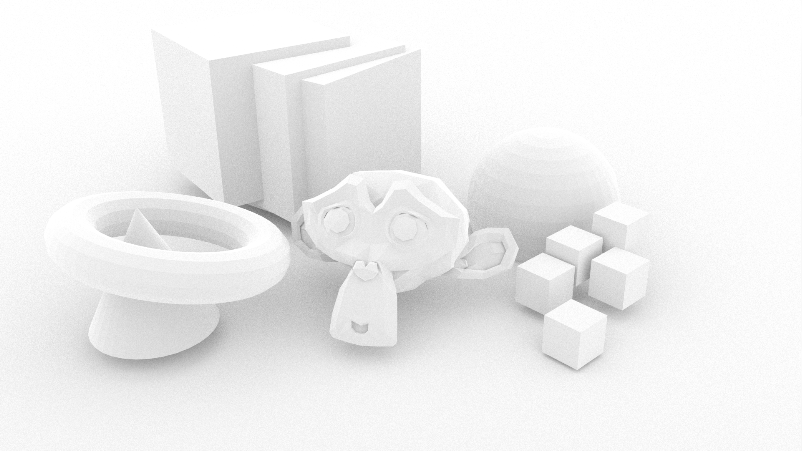
# Ambient occlusion

# Ambient occlusion

**Ambient occlusion** (AO) is a factor determining the amount of ambient light at the scene point.

This occlusion is due to geometric **obstacles** in the point's neighborhood, **blocking** some fraction of incoming ambient light from the surrounding environment.

# Ambient occlusion - Example



Raytraced ambient occlusion of a scene.
(rendered in Blender over ∼20 seconds).

# Ambient occlusion (2)

Ambient occlusion is a step towards **global illumination**.

Increases visual quality, enhances shading of small **details** and improves perception of **spatial distribution** of objects in the scene.

# Ambient occlusion (2)

Ambient occlusion is a step towards **global illumination**.

Increases visual quality, enhances shading of small **details** and improves perception of **spatial distribution** of objects in the scene.

Ambient occlusion term is **view independent**.
For static scenes can be precomputed and stored in textures.
Ambient maps can be used in real-time applications.

# Sub-surface scattering

# SSS via texture-space diffusion

For real-time simulation of the sub-surface scattering we use a technique proposed by d'Eon et al. [1].

# SSS via texture-space diffusion

For real-time simulation of the sub-surface scattering we use a technique proposed by d'Eon et al. [1].

Method can be split into **three stages**:

# SSS via texture-space diffusion

For real-time simulation of the sub-surface scattering we use a technique proposed by d'Eon et al. [1].

Method can be split into **three stages**:

- Geometry pass: rendering to texture-space

# SSS via texture-space diffusion

For real-time simulation of the sub-surface scattering we use a technique proposed by d'Eon et al. [1].

Method can be split into **three stages**:

- Geometry pass: rendering to texture-space
- Fragment shader pass: shading and blurring

# SSS via texture-space diffusion

For real-time simulation of the sub-surface scattering we use a technique proposed by d'Eon et al. [1].

Method can be split into **three stages**:

- Geometry pass: rendering to texture-space
- Fragment shader pass: shading and blurring
- Geometry pass: final composition

# SSS via texture-space diffusion: Phase 1

In **Phase 1** we map each vertex to its texture coordinates:

# SSS via texture-space diffusion: Phase 1

In **Phase 1** we map each vertex to its texture coordinates:

$$\underbrace{\begin{bmatrix} \text{Position} & (x, y, z, w) \\ \text{Color} & (r, g, b, a) \\ \text{TexCoords} & (u, v) \\ \text{Normal} & (n_x, n_y, n_z) \\ \text{Tangent} & (t_x, t_y, t_z) \\ & \vdots \end{bmatrix}}_{\text{Vertex data}}$$

# SSS via texture-space diffusion: Phase 1

In **Phase 1** we map each vertex to its texture coordinates:

$$\underbrace{\begin{bmatrix} \text{Position} & (x, y, z, w) \\ \text{Color} & (r, g, b, a) \\ \textbf{TexCoords} & (\mathbf{u}, \mathbf{v}) \\ \text{Normal} & (n_x, n_y, n_z) \\ \text{Tangent} & (t_x, t_y, t_z) \\ & \vdots \end{bmatrix}}_{\text{Vertex data}} \mapsto (u, v)$$

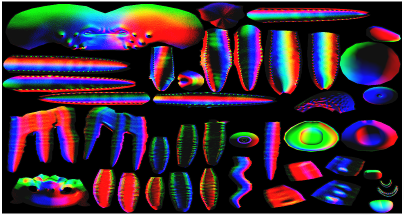
# SSS via texture-space diffusion: Phase 1

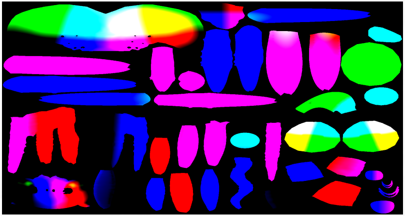
In **Phase 1** we map each vertex to its texture coordinates:

$$\underbrace{\begin{bmatrix} \text{Position} & (x,y,z,w) \\ \text{Color} & (r,g,b,a) \\ \textbf{TexCoords} & (\mathbf{u},\mathbf{v}) \\ \text{Normal} & (n_x,n_y,n_z) \\ \text{Tangent} & (t_x,t_y,t_z) \\ & \vdots \end{bmatrix}}_{\text{Vertex data}} \mapsto (u,v)$$

This way the mesh is **unwrapped** into 2D plane.

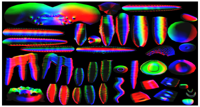
# SSS via texture-space diffusion: Phase 1

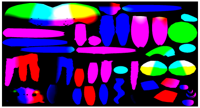




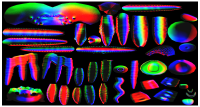
Left: world-space normal. Right: world-space position.

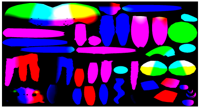
# SSS via texture-space diffusion: Phase 2

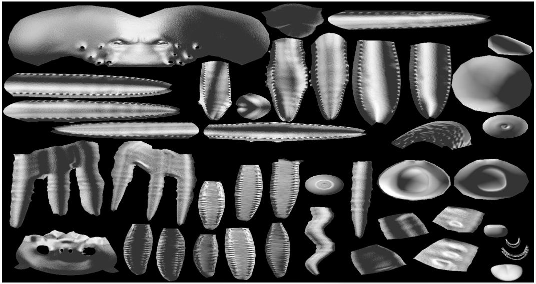




# SSS via texture-space diffusion: Phase 2

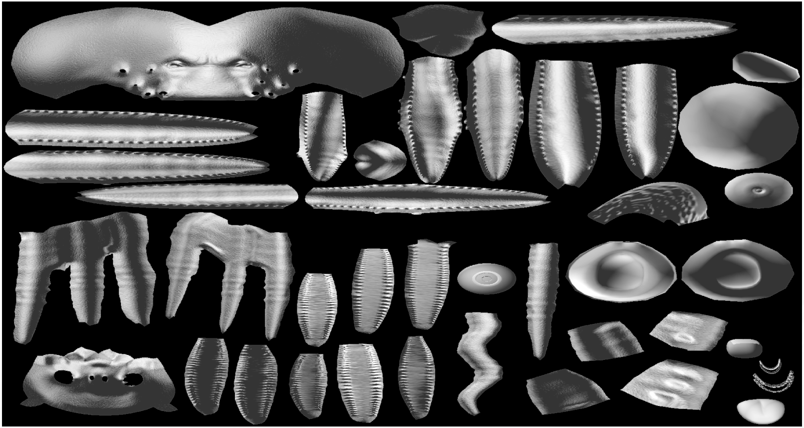






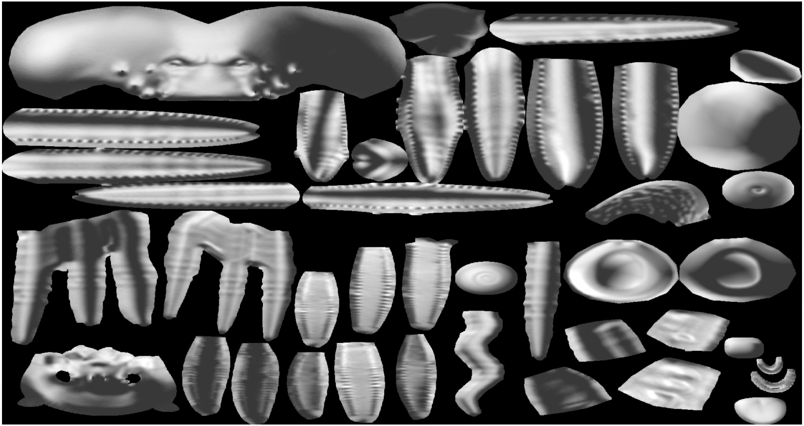
Bottom image: a lightmap computed from texture-space information collected in the previous phase.

# SSS via texture-space diffusion: Phase 2



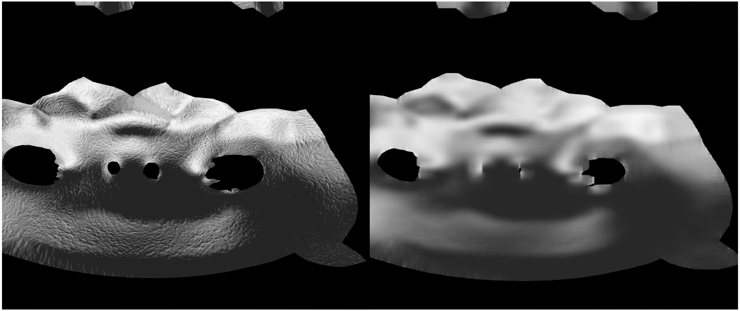
Lightmap containing ambient and diffuse Phong components.

# SSS via texture-space diffusion: Phase 2



Blurring the texture by a separable gaussian blur.

# SSS via texture-space diffusion: Phase 2



Blur detail. Note inflated contours.

# SSS via texture-space diffusion: Phase 3

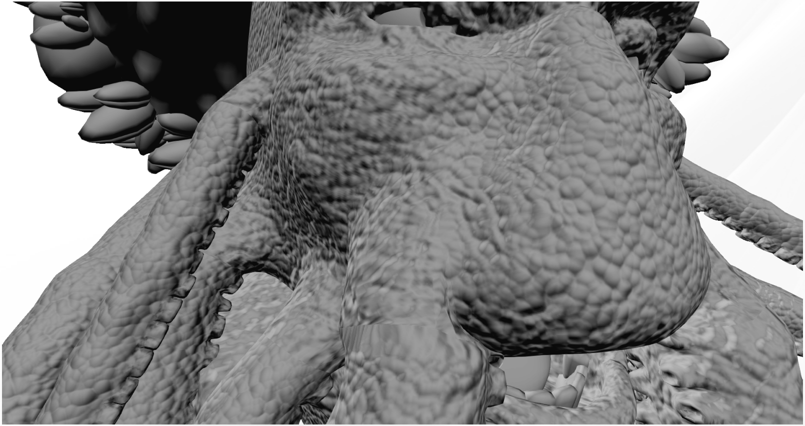
In the **third phase** the blurred texture is mapped to the model.

# SSS via texture-space diffusion: Phase 3

In the **third phase** the blurred texture is mapped to the model.

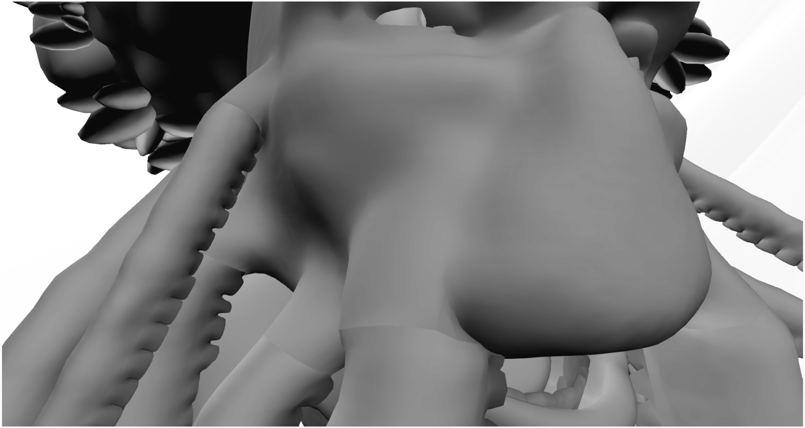
Since the lightmap only contains ambient and diffuse light, we add **specular** highlights separately.

# SSS via texture-space diffusion: Phase 3



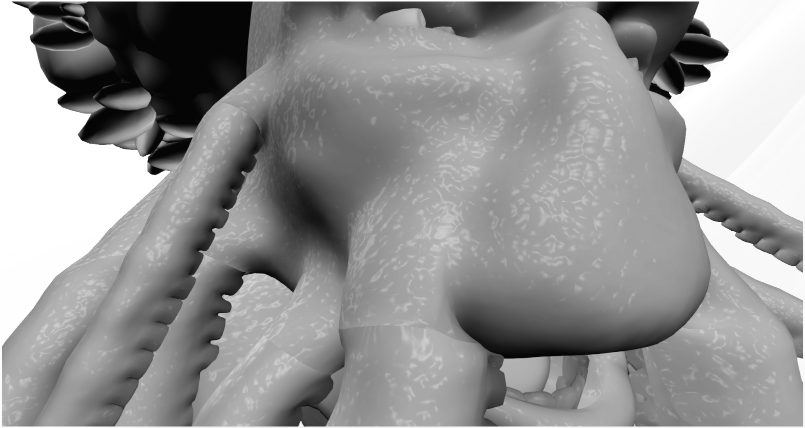
Lightmap (ambient + diffuse)

# SSS via texture-space diffusion: Phase 3



Blurred lightmap (13x13tx)

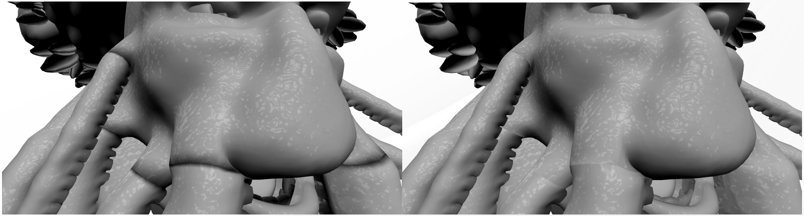
# SSS via texture-space diffusion: Phase 3



Blurred light with added highlights

# SSS via texture-space diffusion: Problem

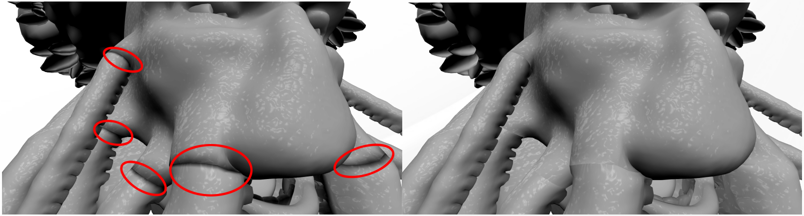
Note that we cannot use arbitrary blurring method, or the **back-ground** may **leak** into shaded areas and create visible seams.



Left: visible leaks of background. Right: fixed.

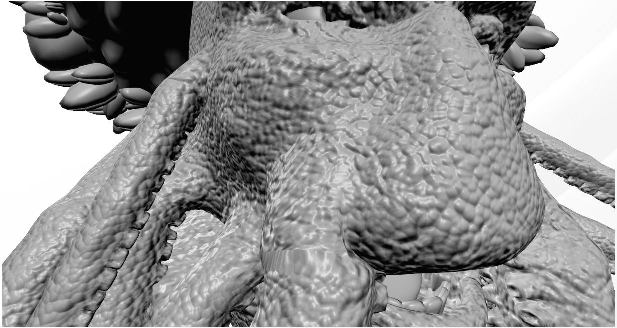
# SSS via texture-space diffusion: Possible complication

Note that we cannot use arbitrary blurring method, or the **background** may **leak** into shaded areas and create visible seams.



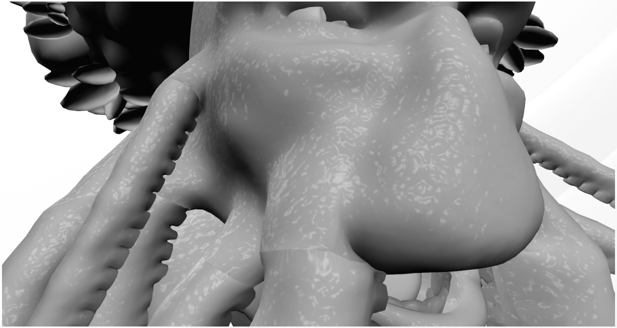
Left: visible leaks of background. Right: fixed.

# SSS via texture-space diffusion: Result



Model WITHOUT the sub-surface scattering effect.

# SSS via texture-space diffusion: Result



Model WITH the sub-surface scattering effect.

# Screen-space ambient occlusion

# Screen-space AO

**Screen-space ambient occlusion** (SSAO) is a real-time method estimating AO term for each screen pixel.

The technique is described in Mittring's article [4] about developing computer game Crysis.

# Screen-space AO

**Screen-space ambient occlusion** (SSAO) is a real-time method estimating AO term for each screen pixel.

The technique is described in Mittring's article [4] about developing computer game Crysis.

Compromise between **speed** and **accuracy**.

**Independent** from the geometric complexity (triangle count and distribution) of the scene.

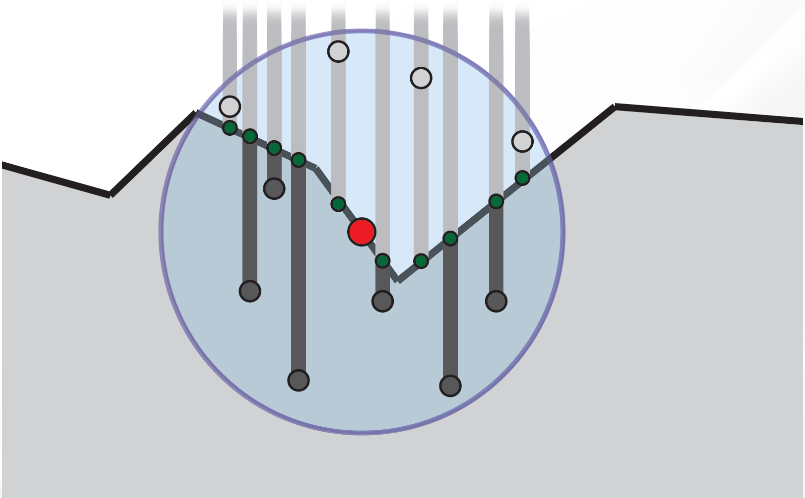
# Screen-space AO (2)

**Sampling** every screen pixel's neighborhood.

Comparing **depth** values (read from z-buffer) of these samples.

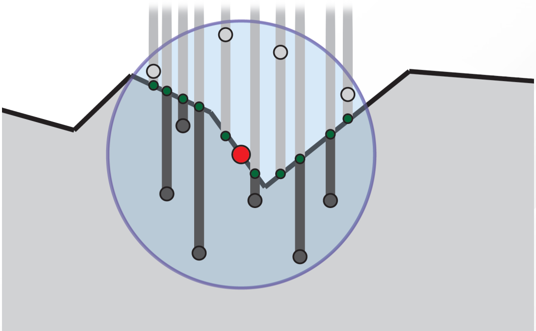
Every sample that falls behind the geometry of the scene contributes to the examined pixel's occlusion.

# Screen-space AO - Sampling



Sampling the red pixel's neighbors.

# Screen-space AO - Sampling



Sampling the red pixel's neighbors.
Gray dots are taken samples, green dots represent values in the z-buffer.

The more the dark samples, the more occluded the red point is.

# Screen-space AO - Parameters

The effect is controlled by two parameters:

# Screen-space AO - Parameters

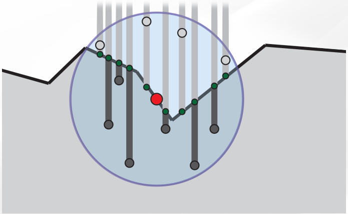
The effect is controlled by two parameters:

- Number of samples taken
- Radius of the sphere around the examined pixel

# Screen-space AO - Parameters

The effect is controlled by two parameters:

- Number of samples taken
- Radius of the sphere around the examined pixel



# Screen-space AO - Parameters

Left to right: radius 12px, 60px, 200px.

Top row: 4 samples/pixel. Bottom row: 64 samples/pixel.

# Screen-space AO - Parameters

Left to right: radius 12px, 60px, 200px.

# Screen-space AO - Merging

Combining **high**, **middle** and **low** frequency maps:

$$\mathrm{SSAO} = \sqrt[a+b+c]{(\mathrm{High})^a \cdot (\mathrm{Medium})^b \cdot (\mathrm{Low})^c}$$

We experimentally found values giving visually best results as $a = 1$, $b = 2.2$ and $c = 3.8$.

# Screen-space AO - Result

Final SSAO map composition compared to raytraced AO (Blender; ∼10s).

# Environment mapping

# Environment mapping

**Environment mapping** (EM) is a fast method for rough estimation of mirror reflections.

Only produces **fake reflections**, but it is awesomely **cheap**.

We are using the technique to achieve wet and slimy look of the rendered surfaces.

# Environment mapping (2)

We use an **environment texture** describing the surroundings of the object.

# Environment mapping (2)

We use an **environment texture** describing the surroundings of the object.

For each screen pixel, we **reflect** the viewer-to-pixel vector from the pixel.

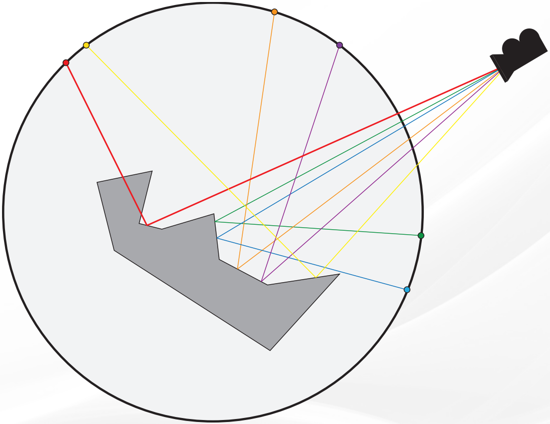
# Environment mapping (2)

We use an **environment texture** describing the surroundings of the object.

For each screen pixel, we **reflect** the viewer-to-pixel vector from the pixel.

The reflected direction determines which texel from the environment texture is to be used as the reflected color.

# Environment mapping - Concept



Concept of environment mapping.

Rays from the camera bounce from the object and hit the background.

# Environment mapping - Result

Environment light reflected from the object.

# Environment mapping - Improvement

Unconvincing results for complex objects due to lack of **multiple reflections**.

# Environment mapping - Improvement

Unconvincing results for complex objects due to lack of **multiple reflections**.

Easy and fast solution uses the **SSAO** map and **multiplies** reflection intensity by the amount of occlusion.

# Environment mapping - Improvement

Unconvincing results for complex objects due to lack of **multiple reflections**.

Easy and fast solution uses the **SSAO** map and **multiplies** reflection intensity by the amount of occlusion.

The more occluded the pixel is, the more reflections the ray has to undergo to reach it.
Therefore, its energy is significantly subdued.

# Environment mapping - Improvement

Left: basic EM. Right: EM enhanced by SSAO.

# Environment mapping - Result

Left: EM enhanced by SSAO. Right: raytraced reflection (Blender; ~5s)

# Parametric model

# Parametric model

Our skin model has a few parameters that can be modified in order to achieve a different look:

# Parametric model

Our skin model has a few parameters that can be modified in order to achieve a different look:

- **bumpiness**
- **reflectivity**
- **translucency**

# Parametric model

Our skin model has a few parameters that can be modified in order to achieve a different look:

- **bumpiness**
- **reflectivity**
- **translucency**

None of these require any preprocessing, can be changed in real-time.

# Parametric model

Our skin model has a few parameters that can be modified in order to achieve a different look:

- **bumpiness**
- **reflectivity**
- **translucency**

None of these require any preprocessing, can be changed in real-time.

Moreover, we have recently introduced a new feature that allows us to perform certain image processing operations on textures in real time.

# Bumpiness

**Bumpiness** parameter controls how featured the details of the bump map should be.

From left bumpiness: 0, 0.75, 3

# Reflectivity

**Reflectivity** parameter controls how much of the environment light reflects itself from object's surface.

From left reflectivity: 0.07, 0.18, 0.35

# Translucency

**Translucency** parameter controls how much the light is blurred on the object's surface.

From left translucency: 0, 1, 9

# Skin generator

# Performance

# Performance

Our skin rendering module has been tested on a Windows 7 Home system with GeForce GTX560 graphics card, Intel Core i7-950 @3.0GHz processor and 12GB of physical memory.

# Performance

Our skin rendering module has been tested on a Windows 7 Home system with GeForce GTX560 graphics card, Intel Core i7-950 @3.0GHz processor and 12GB of physical memory.

With all effects on, resolution 1920x1080 and with no multisampling, we achieve average framerate **above 20fps**.

# Performance

Our skin rendering module has been tested on a Windows 7 Home system with GeForce GTX560 graphics card, Intel Core i7-950 @3.0GHz processor and 12GB of physical memory.

With all effects on, resolution 1920x1080 and with no multisampling, we achieve average framerate **above 20fps**.

We use many screen-space and texture-space techniques. Therefore the rendering time strongly depends on **screen** and texture **resolutions**.

# Performance - Table

| Mesh complexity | Screen resolution | Framerate |
|---|---|---|
| 65 K | 640 x 480 | 63 |
| 65 K | 800 x 600 | 63 |
| 65 K | 1280 x 720 | 47 |
| 65 K | 1920 x 1080 | 31 |
| 259 K | 640 x 480 | 46 |
| 259 K | 800 x 600 | 35 |
| 259 K | 1280 x 720 | 31 |
| 259 K | 1920 x 1080 | 21 |

Performance of our solution. First column is triangle count, third is average number of frames rendered in a second.

# Results

# Final results

# Final results (2)

# Final results (3)

# Video

Time for a short video...

# Thank You for attention!

Eugene d'Eon, David P. Luebke, and Eric Enderton.
Efficient rendering of human skin.
In Kautz and Pattanaik [3], pages 147–157.

Craig Donner and Henrik Wann Jensen.
Rendering translucent materials using photon diffusion.
In Kautz and Pattanaik [3], pages 243–251.

Jan Kautz and Sumanta N. Pattanaik, editors.
*Proceedings of the Eurographics Symposium on Rendering Techniques, Grenoble, France, 2007*. Eurographics Association, 2007.

Martin Mittring.
Finding next gen: Cryengine 2.
In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121, New York, NY, USA, 2007. ACM.

Michal Valient.

Deferred rendering in killzone 2.
Online, accessed Feb. 20th, 2012, 2007.
Develop Conference, http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf.