Deus Ex is in the Details

Augmenting the PC graphics of Deus Ex: Human Revolution using DirectX 11 technology

Matthijs De Smedt
Graphics Programmer, Nixxes Software

GAME DEVELOPERS CONFERENCE 2012
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9

Overview

- Introduction
- DirectX 11 implementation
- DirectX 11 effects

Nixxes

- Founded in 1999
- Co-development with studios
  - Crystal Dynamics
  - IO Interactive
  - Eidos-Montreal

Deus Ex: Human Revolution

- Developed by Eidos-Montreal
- Nixxes assisted with core technology
- PC version co-developed with Nixxes
- Simultaneous release in August 2011

DXHR Rendering

- Evolved Tomb Raider: Underworld engine
- New rendering features:
  - Hybrid forward lighting and light pre-pass
  - Improved multithreading
  - Artist-controlled postprocessing system

DXHR PC Rendering Goals

- Content creation targeting consoles
- Much more GPU power available on PC
- Minimal artist time available
- How to utilize GPU’s?

PC Rendering Advantages

- New DirectX 11 features
  - Shader Model 5
  - Compute shaders
  - Tessellation
- Improve existing effects


Implementing DirectX 11

DirectX 11 Implementation

- Simple code compared to DX9
- Guaranteed features and formats
- No more device lost
- New API: No experience with optimization

DirectX 11 GPU Optimization

- Read-only depth buffers
- Compute shader local storage
- Gather instruction

DirectX 11 CPU Optimization

- Early on CPU was the bottleneck
- Many unique objects in scene
- Very flexible material system
- Too many state changes

DirectX 11 CPU Optimization

- Minimize state changes between drawcalls
  - Instancing
  - State Objects
  - Constant Buffers
  - Pool static vertex and index buffers

State Objects

- Bound to persistent objects
  - BlendState in Material
- JIT creation and caching in hash tables
  - Hash creation parameters
  - More efficient than Create…State
  - Creation still takes time, you might want to prewarm state objects during startup

Constant buffers

- Bound to objects
  - Light state (for forward lighting)
  - Material parameters
  - Instance parameters
- Other constants split by update frequency
  - Drawable
  - Scene


DirectX 11 Effects

DirectX 11 effects

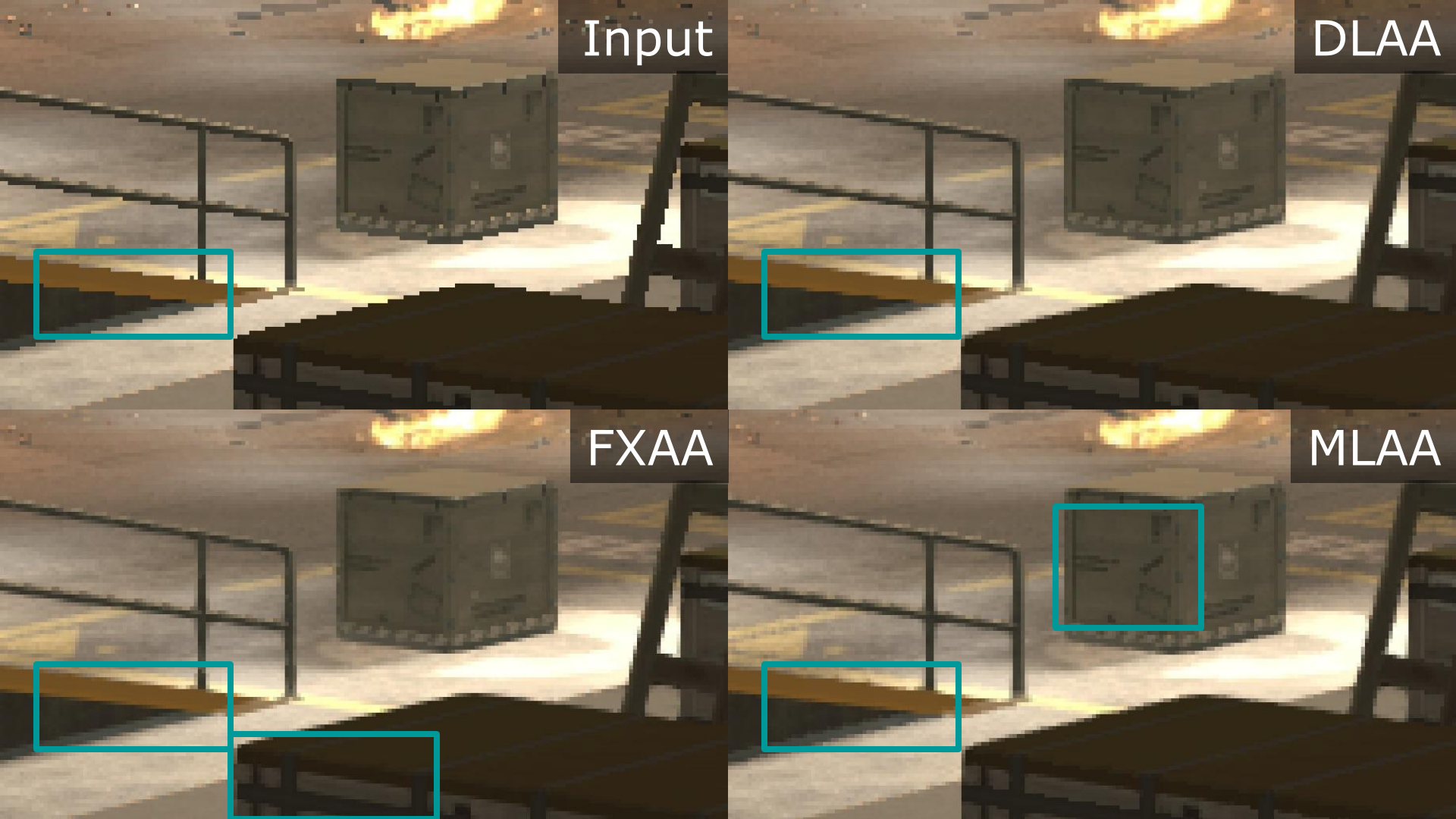
- Anti-aliasing
- SSAO
- Depth of Field
- Tessellation
- Soft shadows

Anti-aliasing

Anti-aliasing

- User preference
  - DLAA
  - FXAA
  - MLAA
- Easy to integrate
- Quality and cost scale


SEA·TEC
PANCHAEA
OLTOSPRO


Input
DLAA
FXAA
MLAA


Depth of Field

Depth of Field

- Experimented with delta spreading
  - Compute shader technique
  - Use atomic ops to achieve point spreading
  - Potential for realistic bokeh
  - Too slow on 2010 hardware

Depth of Field

- DX9: PS Gaussian blur
  - 9x9 filter kernel
- DX11: CS weighted Gaussian blur
  - Separable
  - 29x29 filter kernel
  - Uses thread group shared memory as cache
  - 128x2 pixels group size


Input
Pixel Shader
Compute Shader
9x9
29x29

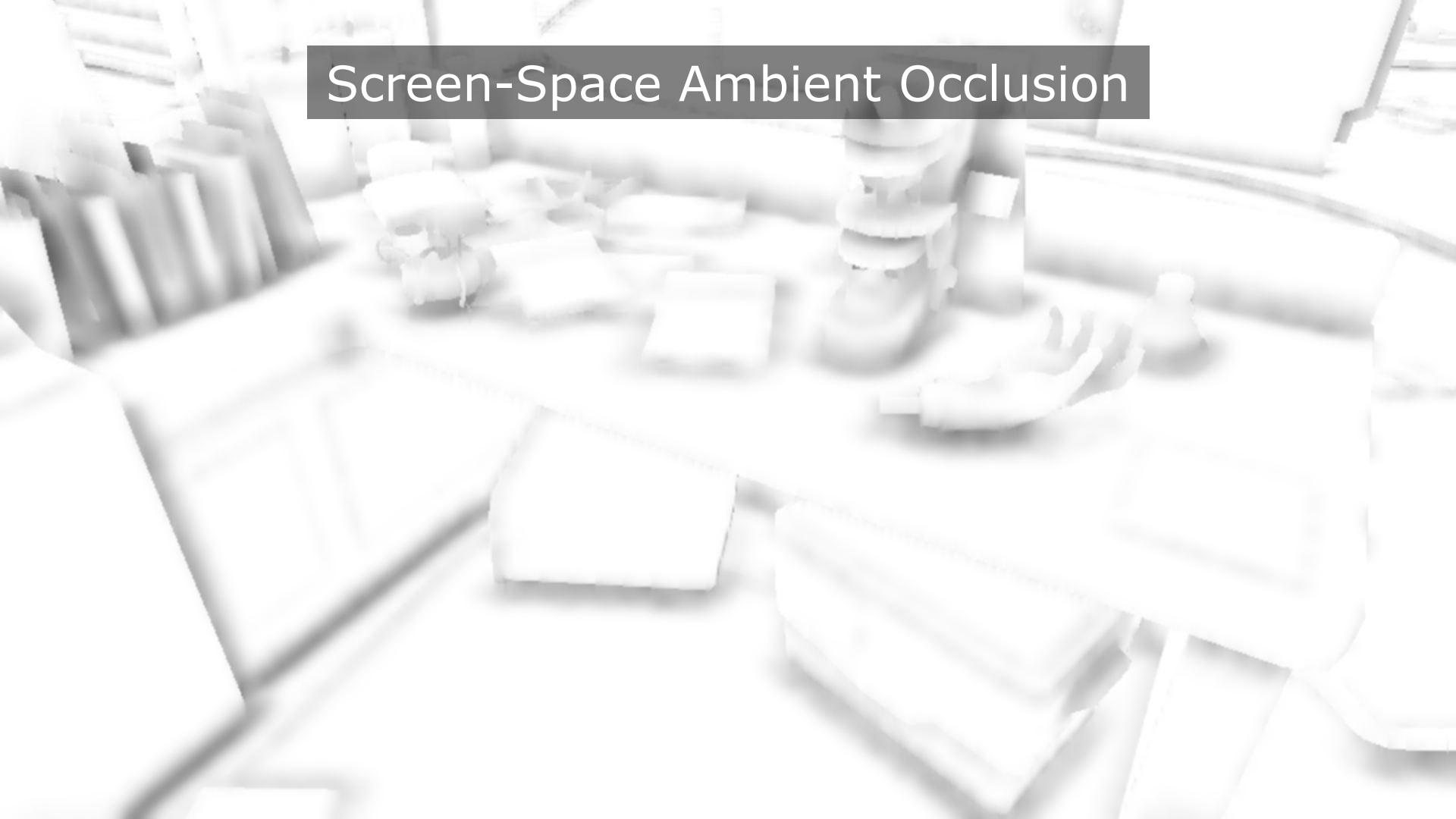
Gaussian Blur Performance

Kernel size	PS	CS	Speedup
9x9	1.17ms	1.25ms	**0.92x**
15x15	1.48ms	1.26ms	**1.17x**
29x29	2.36ms	1.51ms	**1.55x**
41x41	3.11ms	1.74ms	**1.79x**

Resolution: 1920x1080 GPU: AMD HD 6970

Unweighted blur of 8bpp RGB

Screen-Space Ambient Occlusion



Screen-Space Ambient Occlusion

- Console SSAO blurs depth
- PC samples in a hemisphere
  - Less artifacts
  - More expensive
  - Similar to Starcraft 2

SSAO bilateral blur

- DX9
  - Pixel Shader with 9x9 kernel
- DX11
  - Separable Compute Shader with 19x19 kernel
  - Much smoother
  - Reduced noise
  - Small performance hit


SSAO Off
DANGER
FLAMMABLE


SSAO On
DANGER
FLAMMABLE


SSAO On

What actually happened

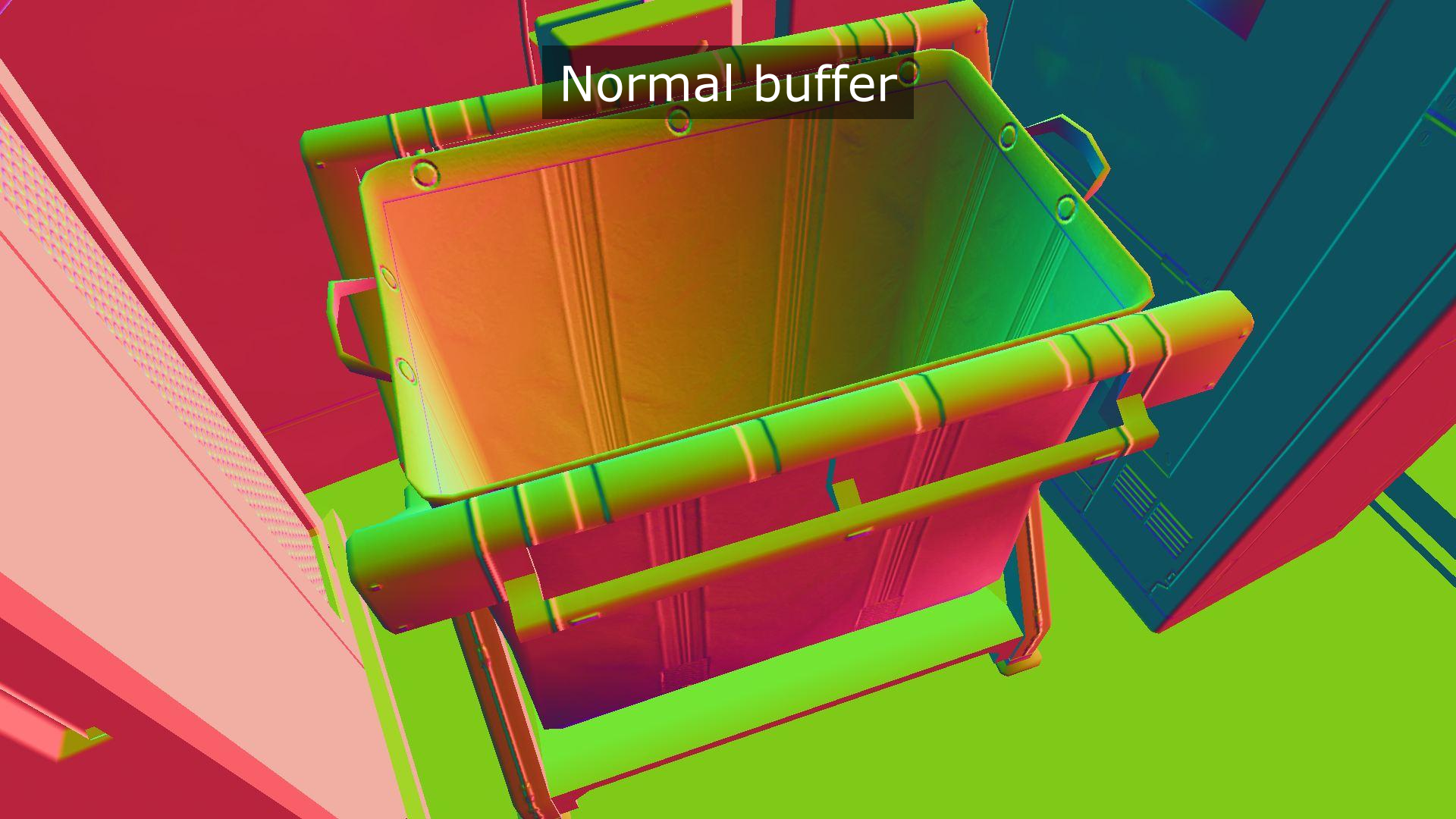
SSAO Self-Occlusion

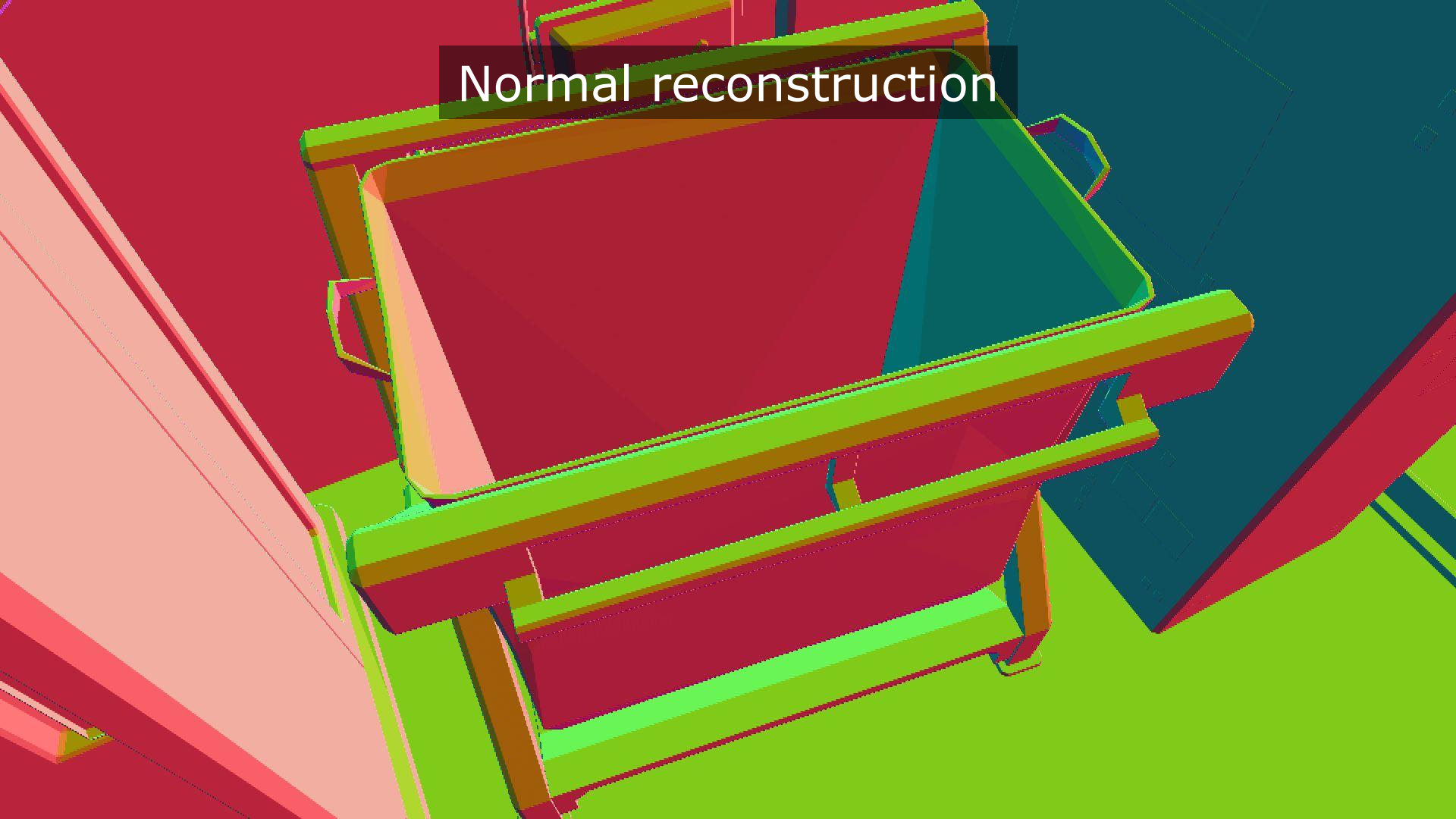
- Problem:
  - Depth buffer is not normal mapped
  - Exaggerated normal maps cause hemispheres to intersect with flat geometry
  - No vertex normals available

SSAO Self-Occlusion

- Solution:
  - Depth buffer contains geometry
  - We want the *viewspace vertex normal*
  - For SSAO we calculate the viewspace position
  - ddx() and ddy() return the slope of any variable
  - Viewspace vertex normal reconstruction:

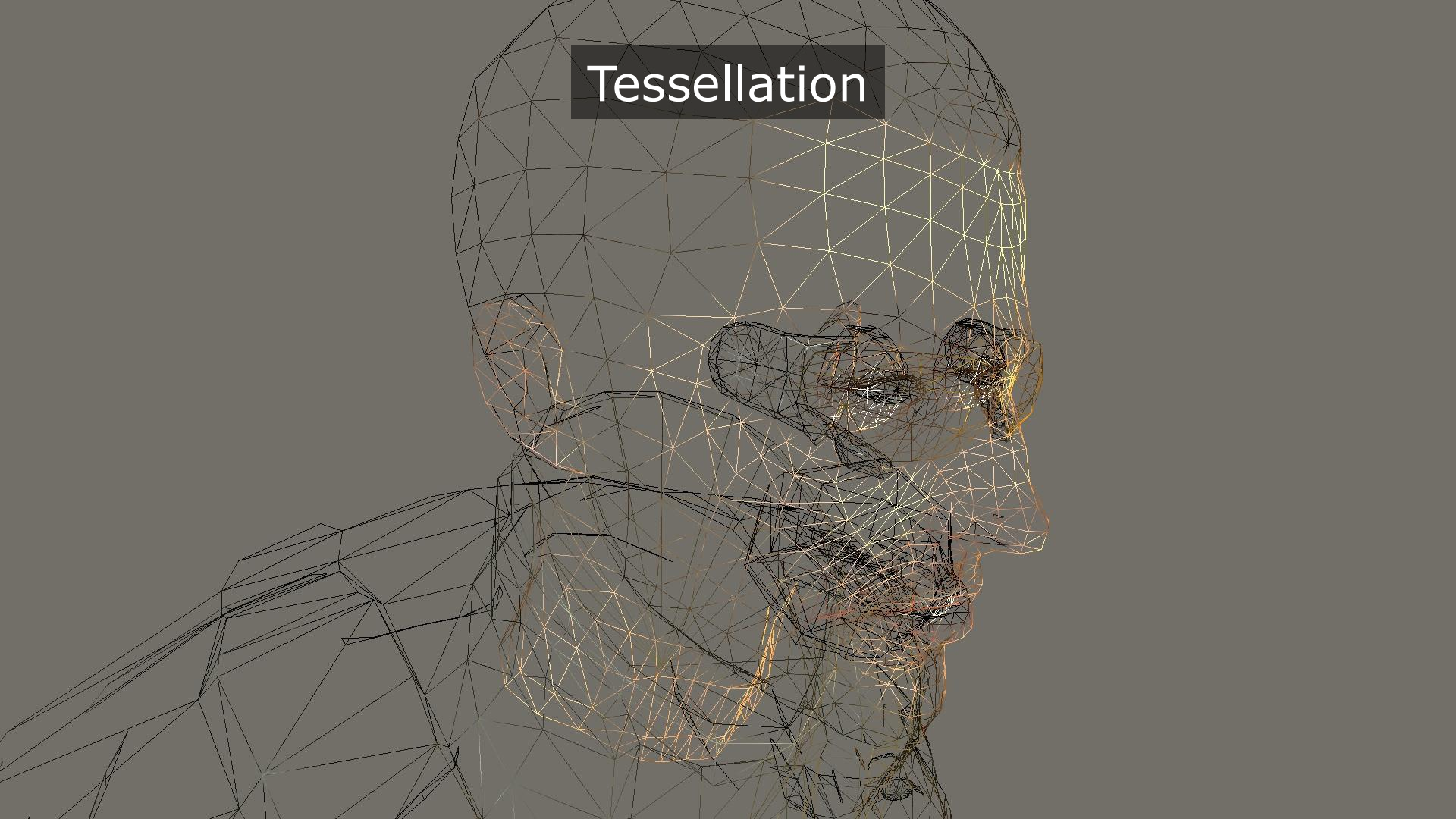
$$\text{normalize}(\text{ddx}(\text{viewpos}) \times \text{ddy}(\text{viewpos}))$$

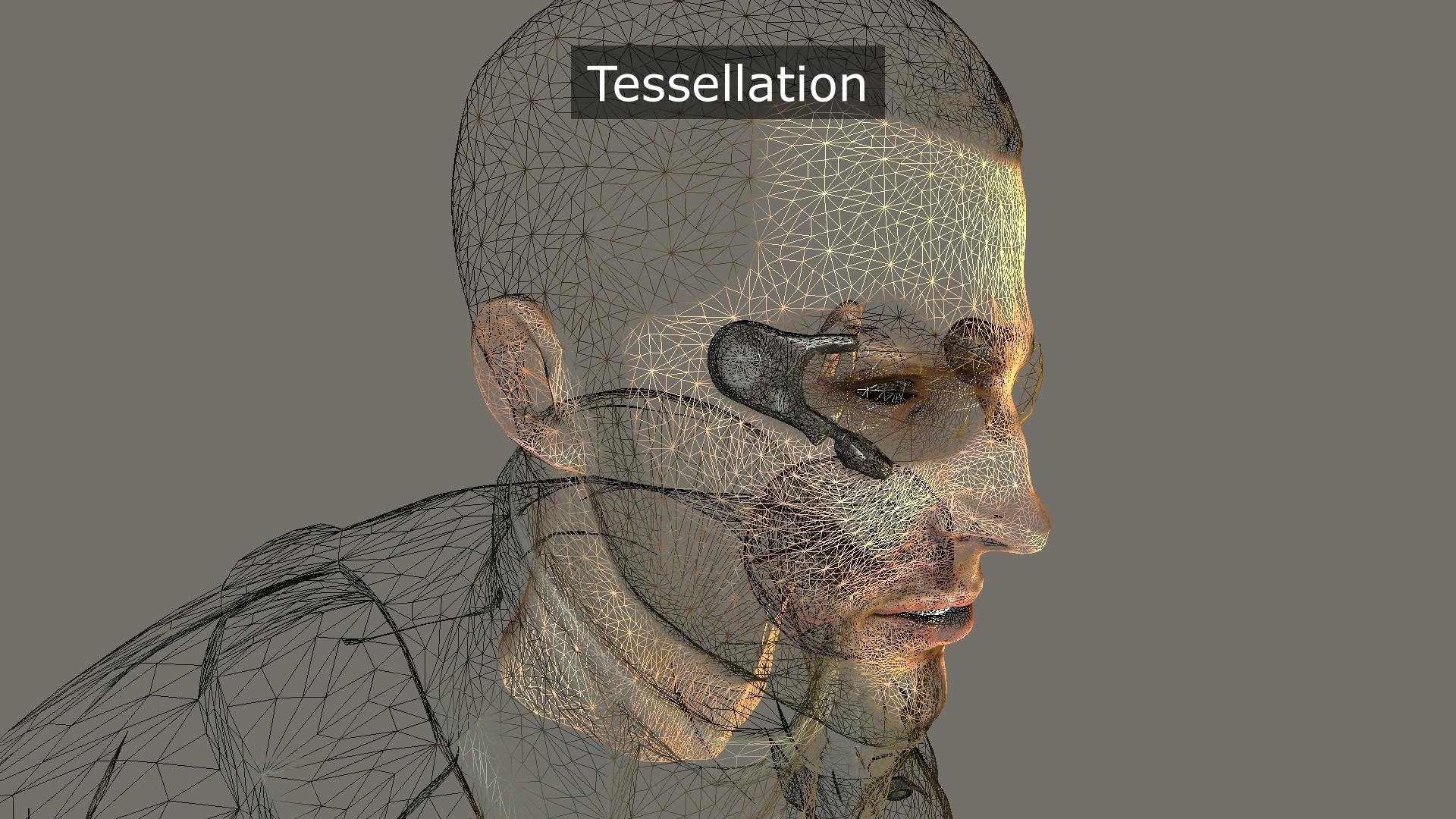

Normal buffer


Normal reconstruction


SSAO On (Fixed)

Tessellation



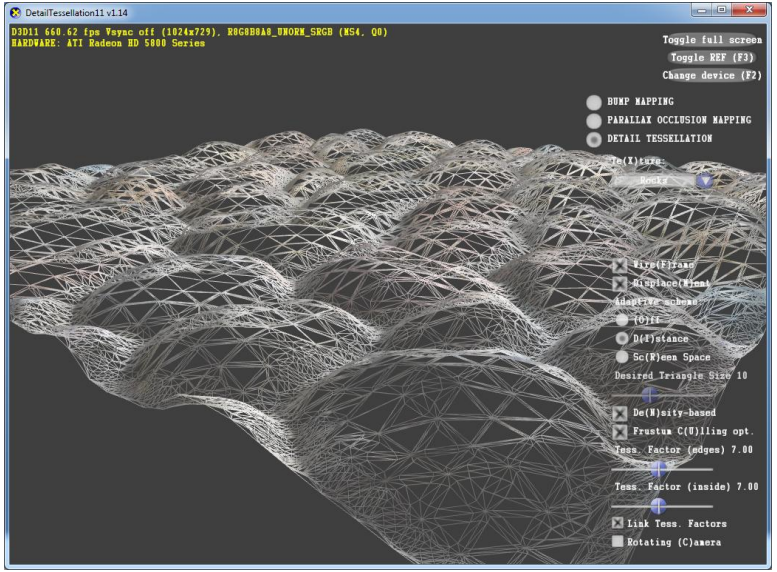

Tessellation

DirectX 11 Tessellation

- New stages supported on all hardware:
  - Hull Shader
  - Domain Shader
- We considered these techniques:
  - Detail Tessellation
  - Geometry smoothing

Detail Tessellation

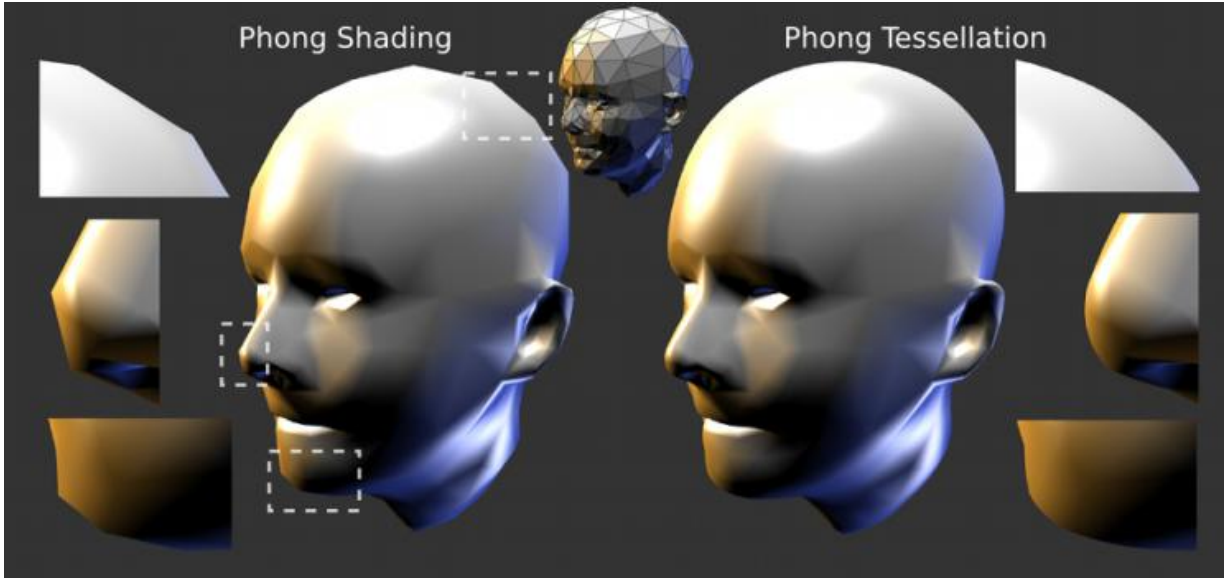
- Tessellate and displace
- Looks great!
- But:
  - Requires height maps
  - Geometry needs to be carefully positioned to avoid cracks



Geometry smoothing

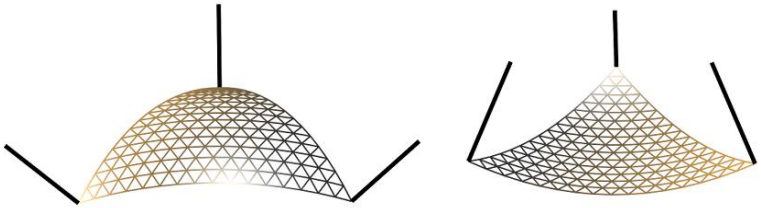
- Smoothes the contours of characters
- Two popular techniques
  - Phong Tessellation
  - PN-Triangles
- Equivalent results but “Phong” is faster

Phong Tessellation



Boubekeur and Alexa, SIGGRAPH Asia 2008

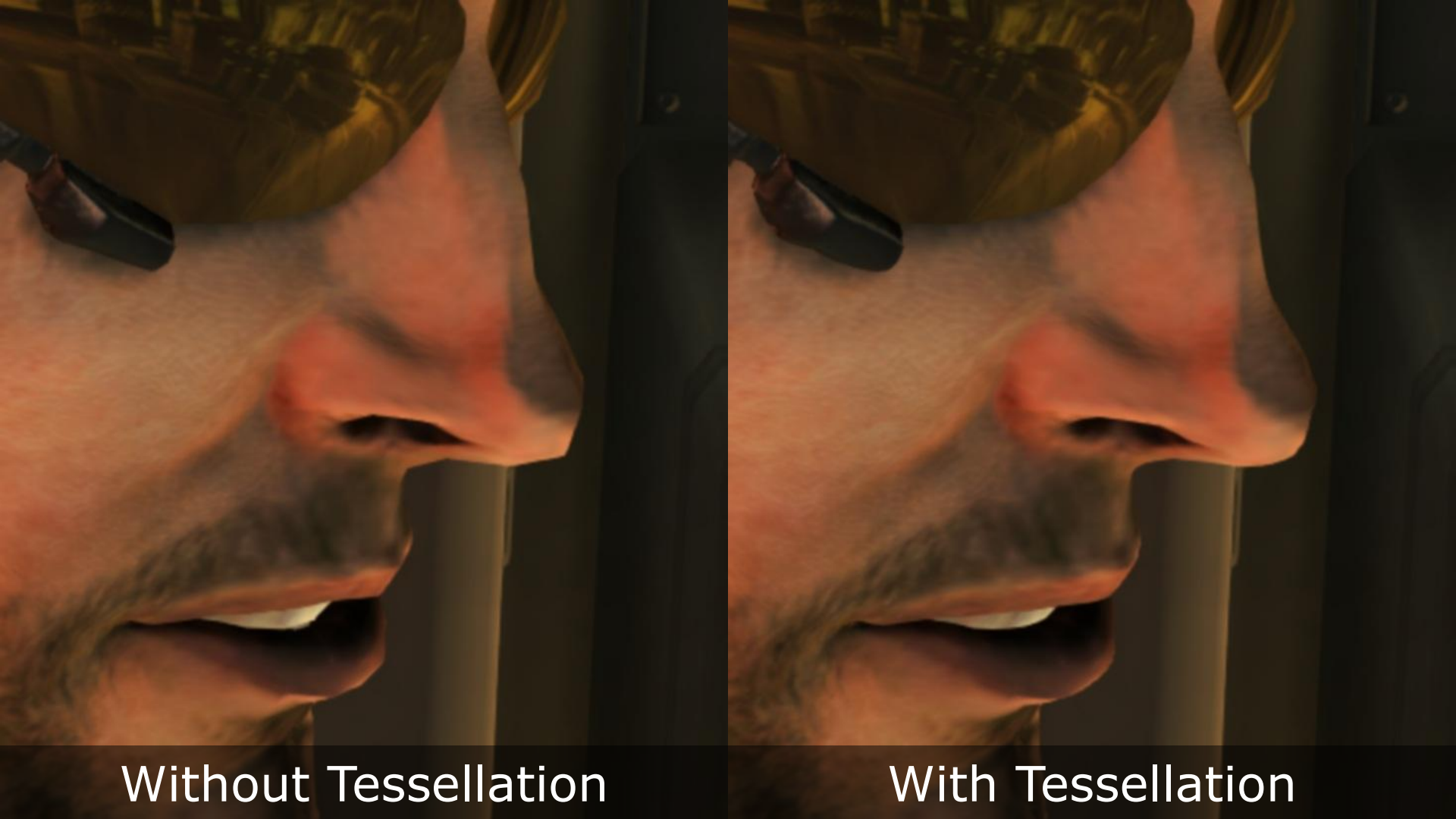
Phong Tessellation

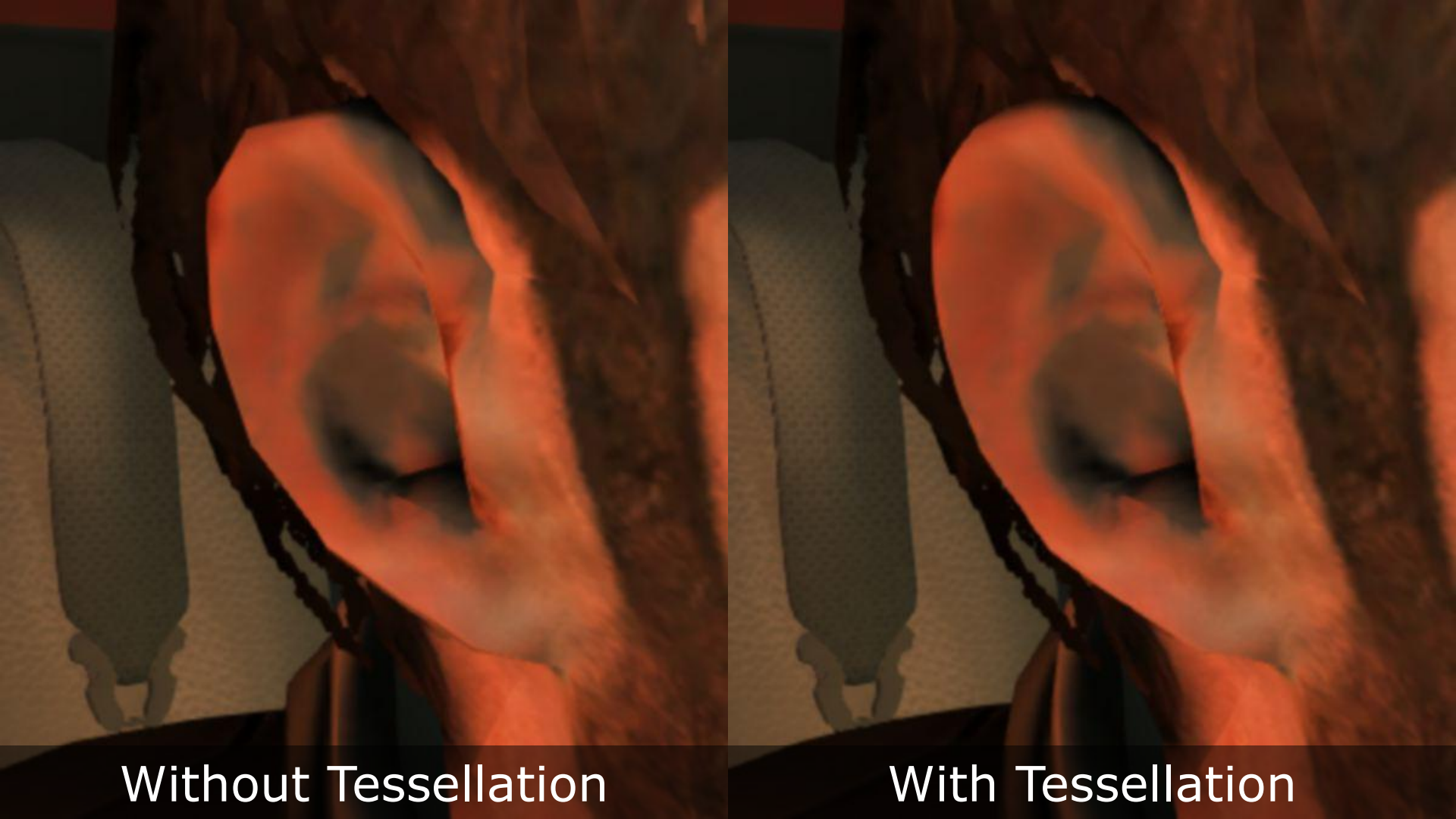


- Simple technique
- Requires only vertex position and normals
- In brief:
  - Calculate tangent plane for each normal
  - Interpolate between tangent planes


Tessellation off


Tessellation on

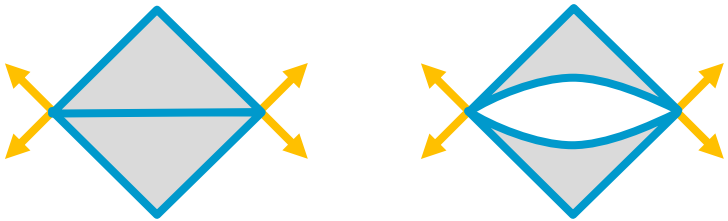

Without Tessellation
With Tessellation


Without Tessellation
With Tessellation


Tessellation Cracks

Tessellation Cracks

- Characters made of multiple submeshes
- Multiple vertices with the same position
- Discontinuous normals



Off On

Tessellation Cracks

- Our solution
  - Generate a “Tessellation Normal” channel
  - Equalizes normal for all verts in this location
  - Normals weighted by triangle size
  - Fixes cracks on mesh boundaries
  - Low overhead

Tessellation Bulges

Tessellation Bulges

- Problem:
  - Hard edges have smooth normals instead
  - This problem appeared in some models
  - Caused by averaging normals to fix cracks!
  - Phong Tessellation creates rounded geometry
- Solution:
  - Artists add extra polygons on edges


POLICE
POLICE
POLICE
POLICE
Original Model
Fixed Model

Tessellation optimizations

- Tessellation is enabled for ~10m distance
- Fade out tessellation before disabling
- Keep the hull shader simple and fast
  - Tessellation factor only distance-based
  - Maximum tessellation factor of 3.0
  - Do cull backfacing triangles with factor 0.0


Soft Shadows

Soft Shadows

- SM5 shader
- 9x9 filter kernel for soft PCF
- Use GatherCmpRed to fetch 4 samples

Soft Shadows

- Problem:
  - All shadow-casting lights rendered with forward lighting
  - 9x9 kernel takes seconds to compile
  - Caused shader build time to explode
  - Too risky to make all lights deferred

Soft Shadows

- Solution:
  - Render soft shadows deferred in screen-space
  - Need only one shader for the game
  - Sample from soft shadow buffer during forward lighting
  - Not used for transparencies

Multi-monitor rendering

- Using vendor-specific API extensions
- You should support all configurations
- How to deal with crosshair in the bezel?
  - Keep original field of view on primary monitor using off-center projection matrix
- Pull back near clip plane when FOV is high
- Increase depth bias for decals

Stereoscopic rendering

- Using vendor-specific API extensions
- Render frame for each eye
- Culling only done once
- Stereoscopic projection matrix

Conclusions

- You can utilize PC GPU's without extra art
- Compute Shaders are great for caching too
- Character tessellation is fast and effective
- We're only getting started using DX11

Special thanks

- Nicolas Thibieroz
- Jon Story
- Miguel Sainz
- Tim Tcheblokov

Thank you for listening!
Any questions?

www.nixxes.com
mdesmedt@nixxes.com