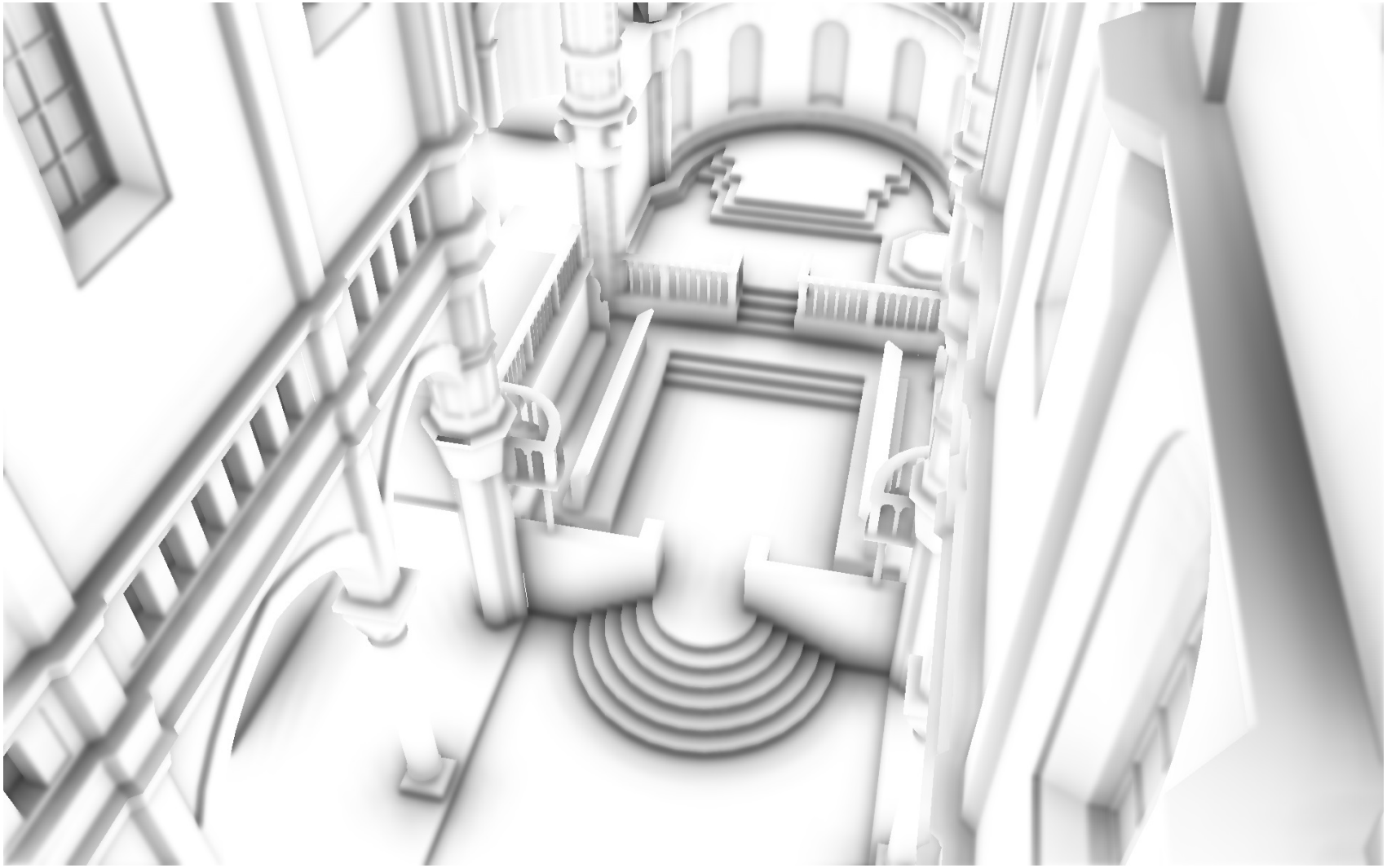
Lesson 5
Screen-space ambient occlusion (SSAO)
Depth of field (DoF)

PV227 – GPU Rendering

Jiří Chmelík, Jan Čejka
Fakulta informatiky Masarykovy univerzity

17. 10. 2016

Ambient occlusion

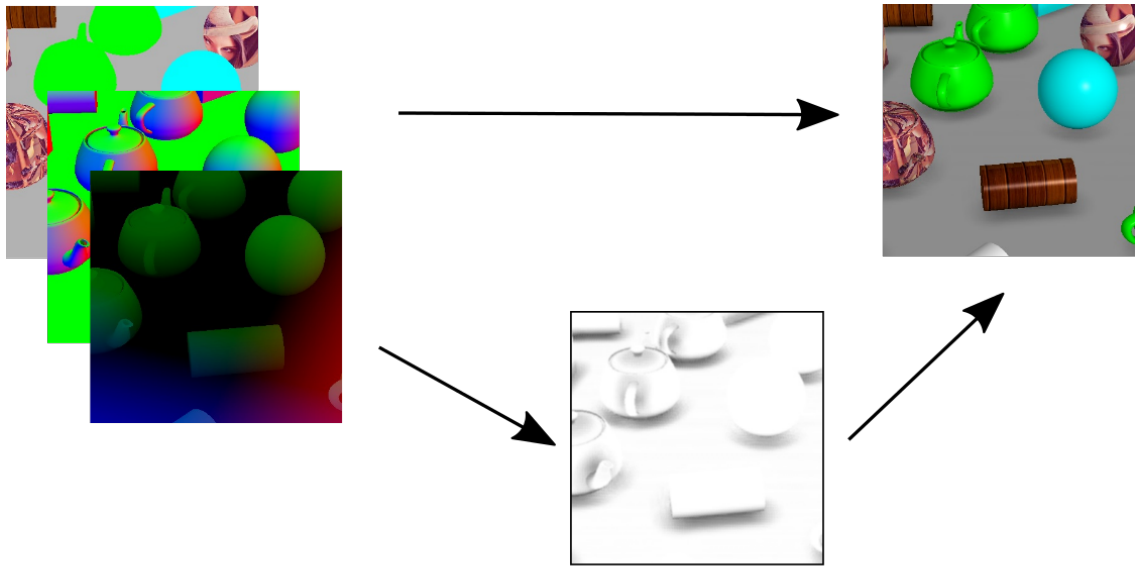


Ambient occlusion (HBAO+ technique, from *GeForce.com*)

Principles of screen-space ambient occlusion

Basic principle

- look at each pixel's neighbourhood and estimate the occlusion



Sampling the neighbourhood

Sampling points in the hemisphere

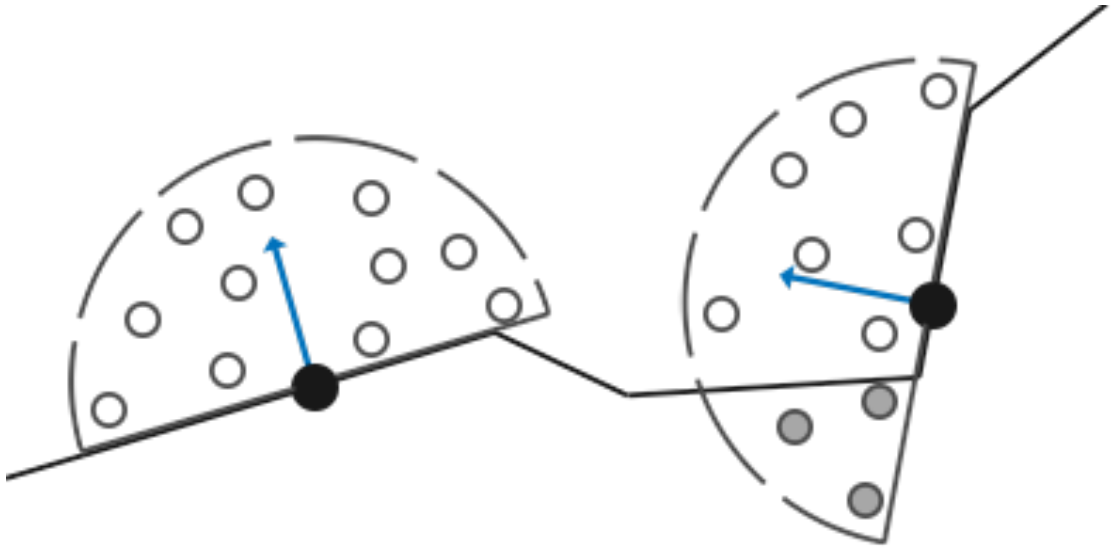
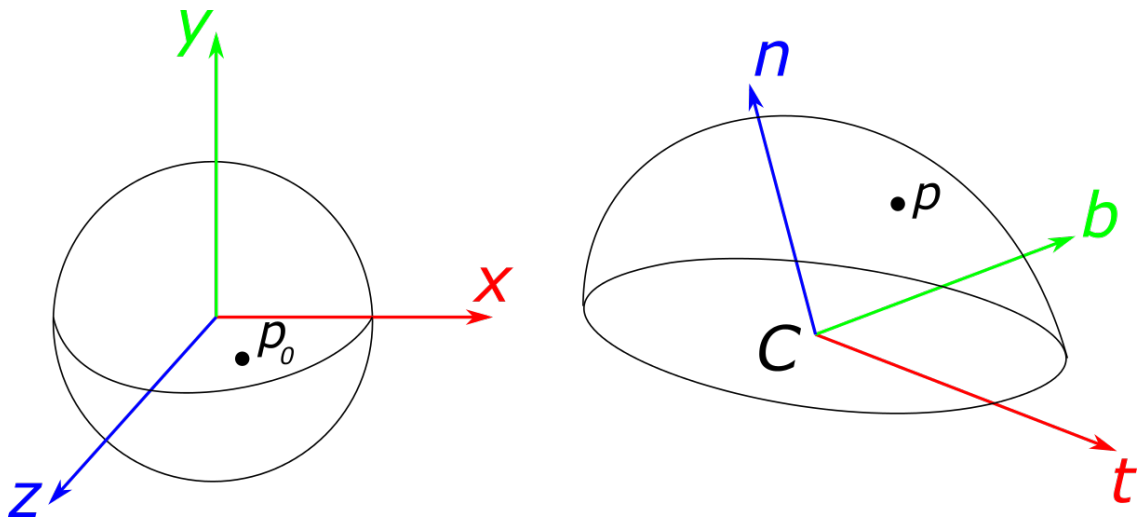


Image from *learnopengl.com*

Tangent space for the hemisphere

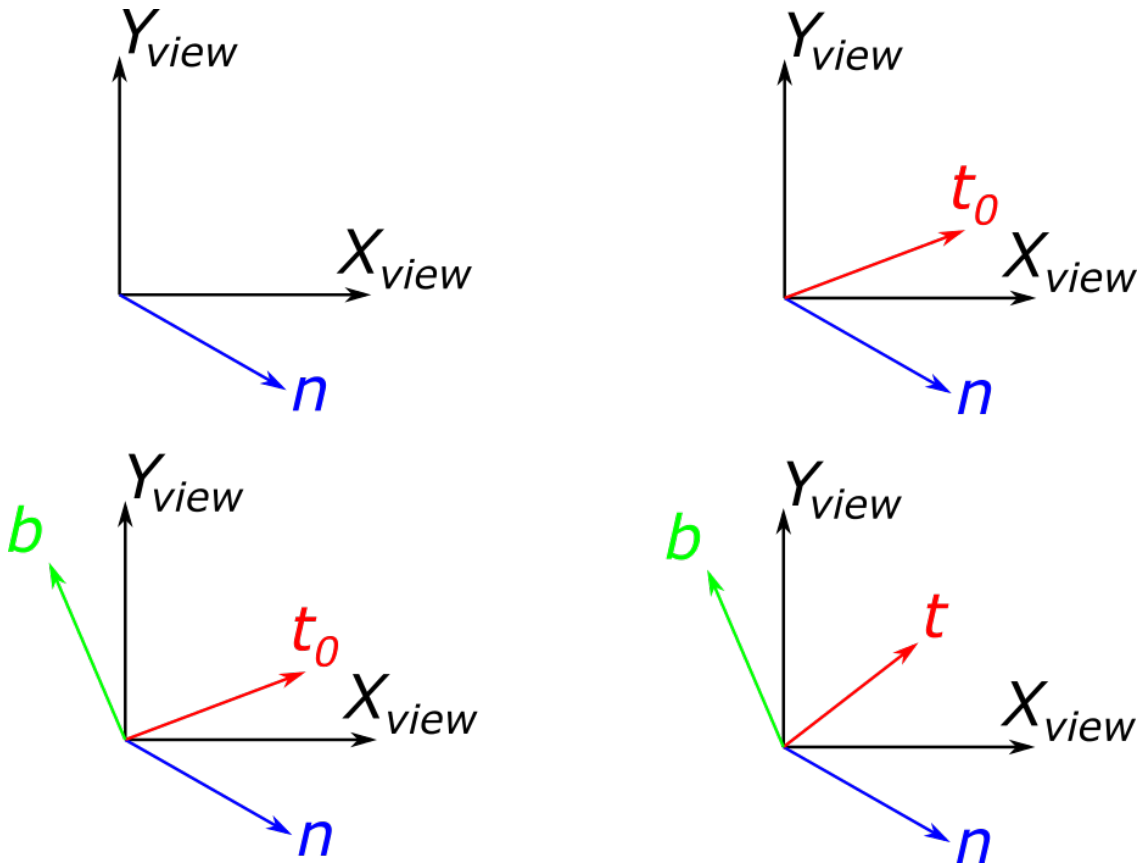
- Basic position $p_0 = (x, y, z)$
- Tangent space $(\vec{t}, \vec{b}, \vec{n})$



$$p = C + x\vec{t} + y\vec{b} + z\vec{n}$$

Finding the tangent and bitangent

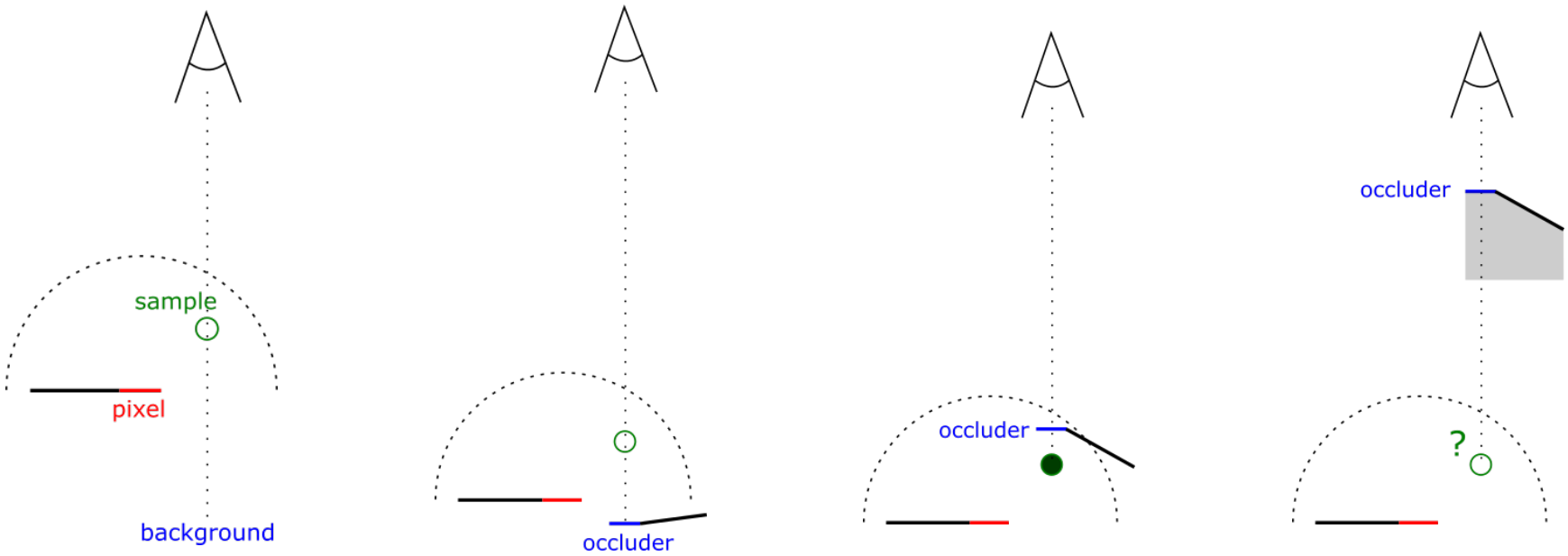
- Choose a random $\vec{t_0}$ in xy plane in view space
- Compute $\vec{b} = \vec{n} \times \vec{t_0}$
- Compute $\vec{t} = \vec{b} \times \vec{n}$



Comparing the distance

Compare the distances from the viewer of:

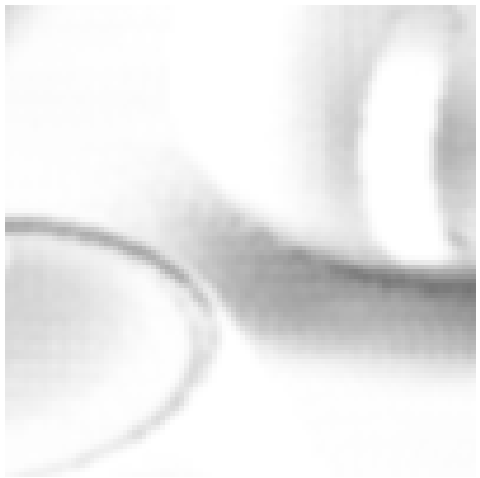
- sample (occludee)
- object visible at that point (occluder)



a) b) c) d)

Blurring the result

Without blur



With blur

Task: Evaluate SSAO

- **Task 1:** Evaluate SSAO
  - Don't worry, a lot of things is already done.
    - ⋆ G-buffer already contains positions and normals in view space.
    - ⋆ Random positions of samples are already computed and stored in a UBO.
    - ⋆ Random directions for the tangents are already computed and stored in a texture.
    - ⋆ Blurring of SSAO texture is already implemented.
    - ⋆ Computation of the tangent and bitangent is already done.
    - ⋆ Computation of the position of the sample and the position of the possible occluder is already done.
  - In *evaluate_ssao_fragment.glsl*, compute the distance of the possible occluder and the sample (occludee) from the camera.
  - Implement comparing cases a), b), and c).

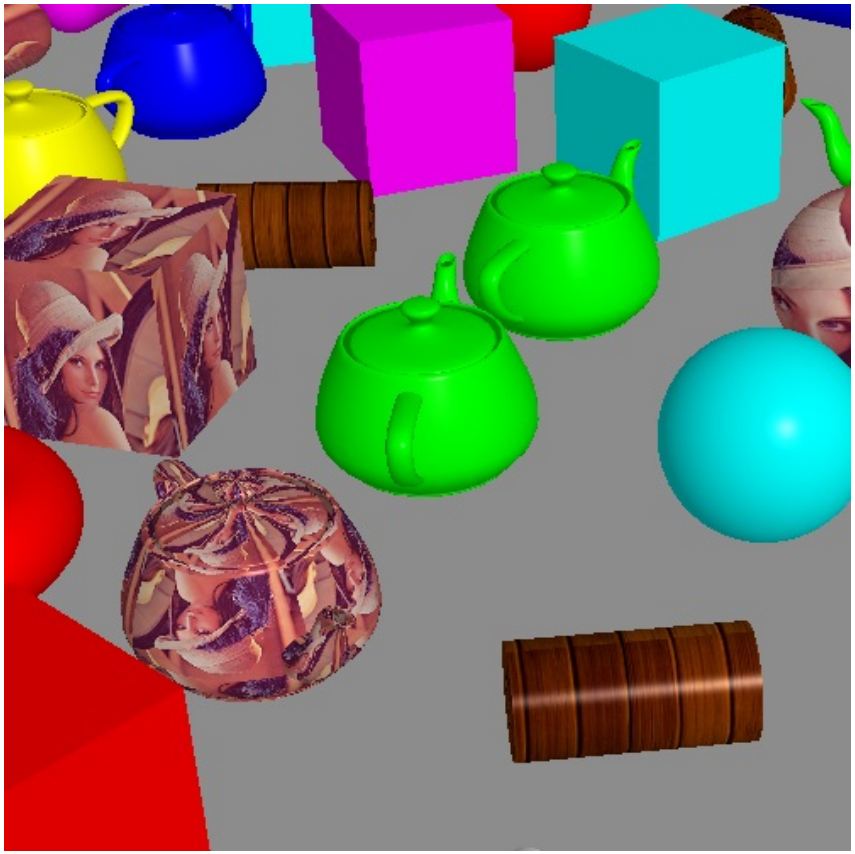
Task: Evaluate SSAO

Result

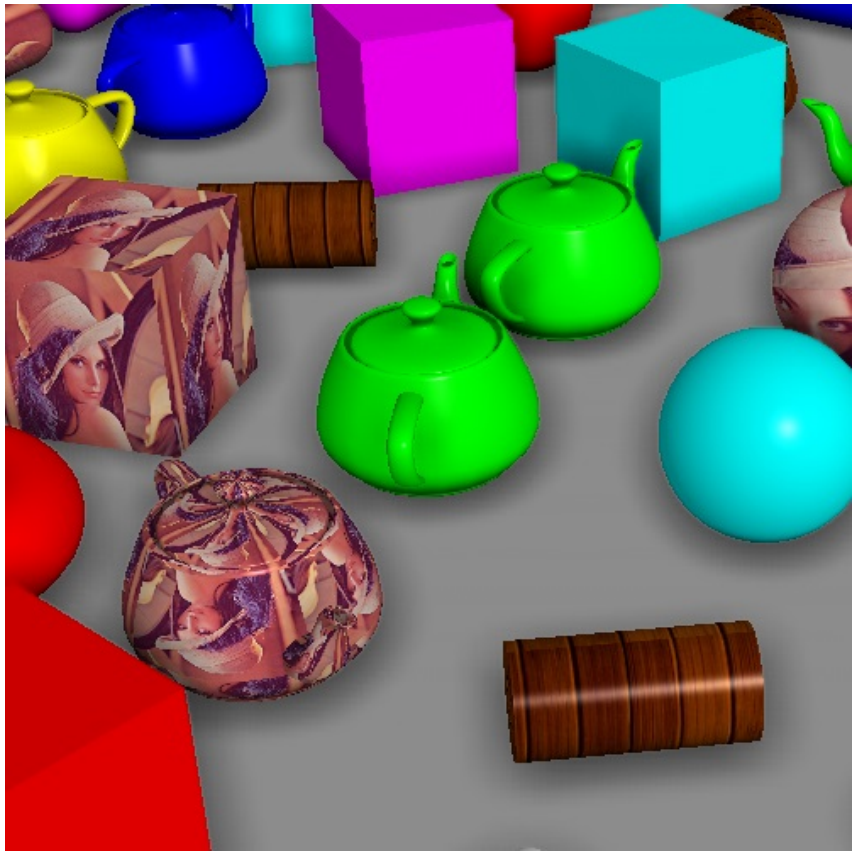
Task: Apply SSAO

- **Task 2:** Apply SSAO to the lighting
  - In *evaluate_lighting_fragment.glsl*, apply the value in texture *ssao_tex* in the computation of lighting.

Task: Apply SSAO



Without SSAO



With SSAO

Task: Improve SSAO

- **Task 3:** Implement comparison case d)

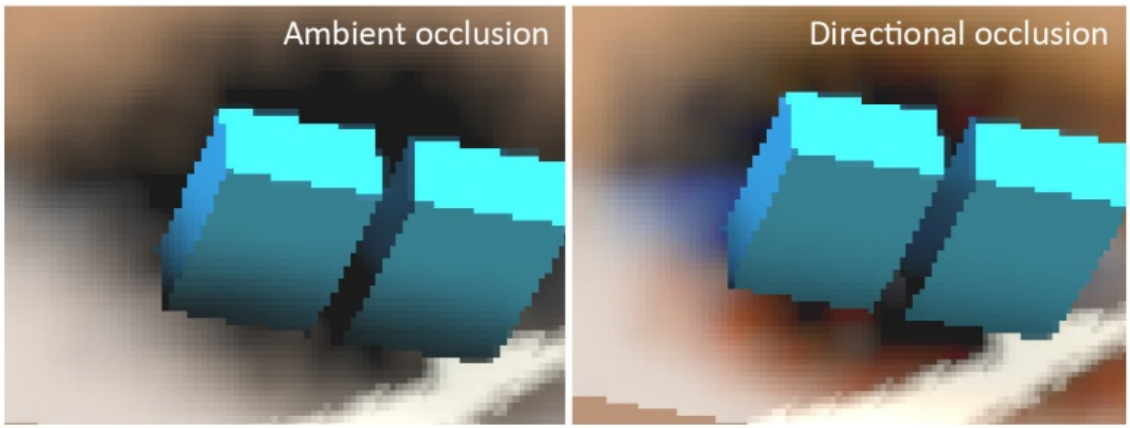
Task: Improve SSAO

Without comparison d)

With comparison d)

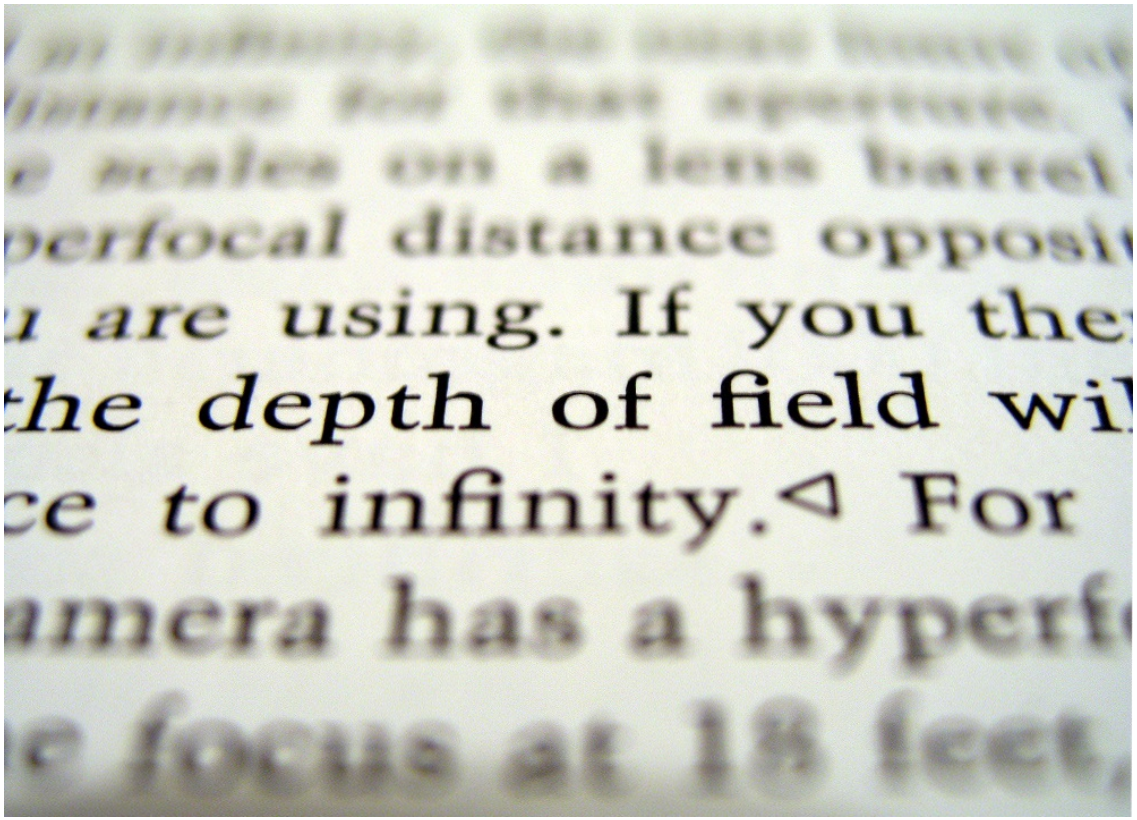
Other methods for ambient occlusion

- Screen-space Directional Occlusion (SSDO)



- Horizon based: HBAO, HBAO+
- Voxels: VXGI

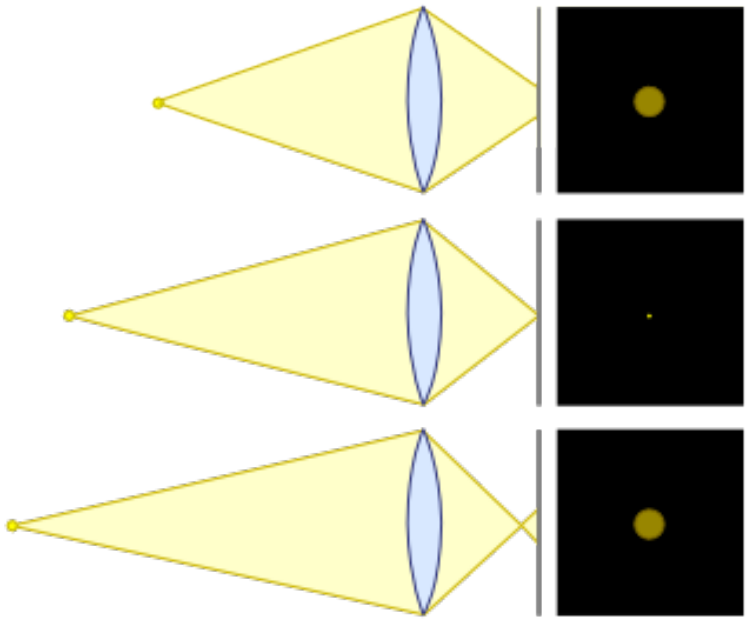
Depth of Field



Depth of field (Source Wikipedia)

Circle of confusion

- Objects out of focus appears as circles (**circle of confusion**)
- Radius of CoC can be calculated from the parameters of the camera

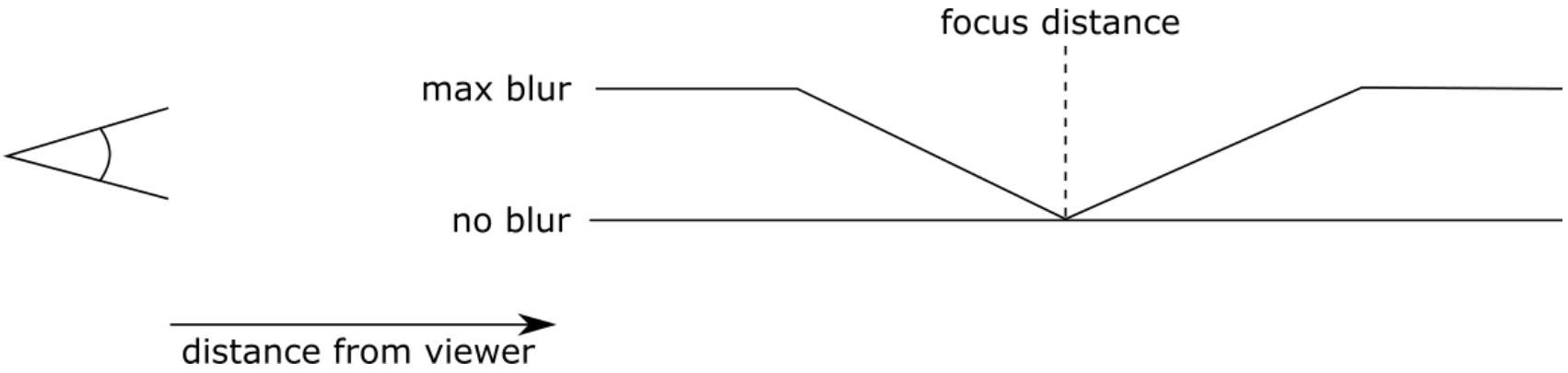


Circles of confusion (Source Wikipedia)

Our simple solution

A simple version:

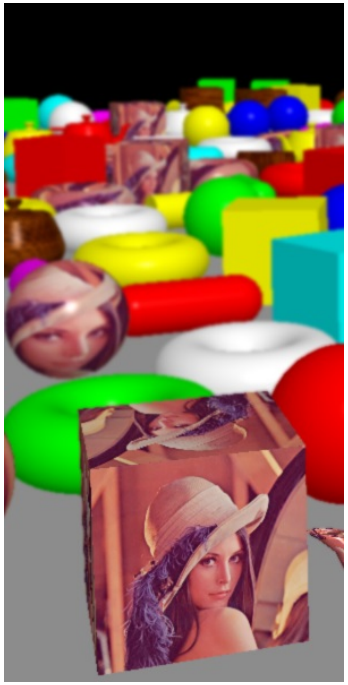
- Linearly increase the blur radius, up to some maximum



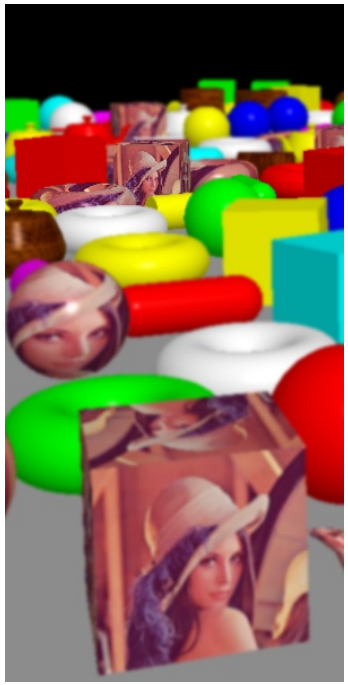
Task: Evaluate Depth of Field

- **Task 4:** Evaluate DoF
  - Shader *dof_fragment.glsl* already blurs the image using gaussian blur.
  - Compute width of the blur and store it into *blur_width*. The distance of objects in focus is in uniform variable *focus_distance*.

Task: Evaluate Depth of Field



Close focus



Middle focus

Far focus

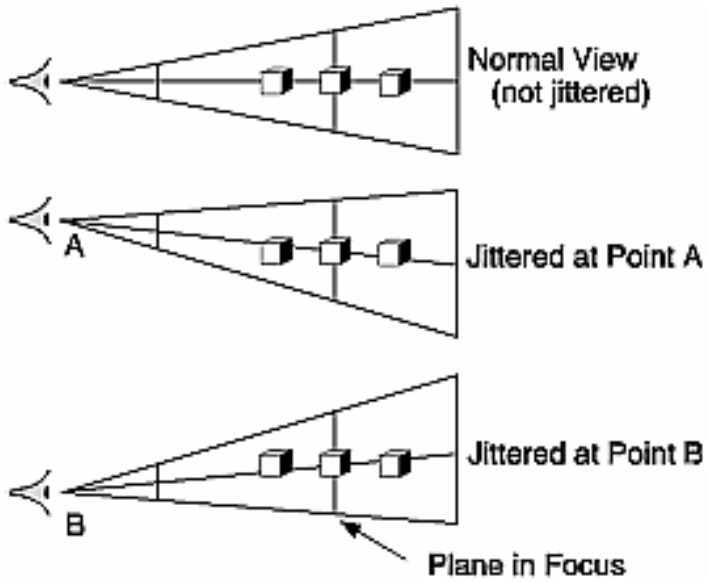
Other methods for DoF

- Splat circles or polygons, one per pixel $\rightarrow$ create bokeh

Bokeh (Source Wikipedia)

Other methods for DoF

- Render the scene multiple times, each time with a slightly shifted camera, and average the results



Source OpenGL Programming Guide

Other methods for DoF

- Render multiple layers, blur each layer uniformly
- In raytracing, simulate the optics of the camera.

Things we used

- *textureLod*
  - Just like *texture*, but explicitely states LOD for mipmap.
  - Necessary when LOD cannot be computed automatically (in divergent code like if's, for's etc.)
- *textureSize*
  - Returns the size of given mipmap level of the texture
- *gl_FragCoord*
  - Only in fragment shaders
  - *.xy* contain the coordinate of the fragment on the screen
  - *.z* contains the depth to be stored into depth buffer
  - *.w* contains $1/w$ of *gl_Position* variable, after interpolation.