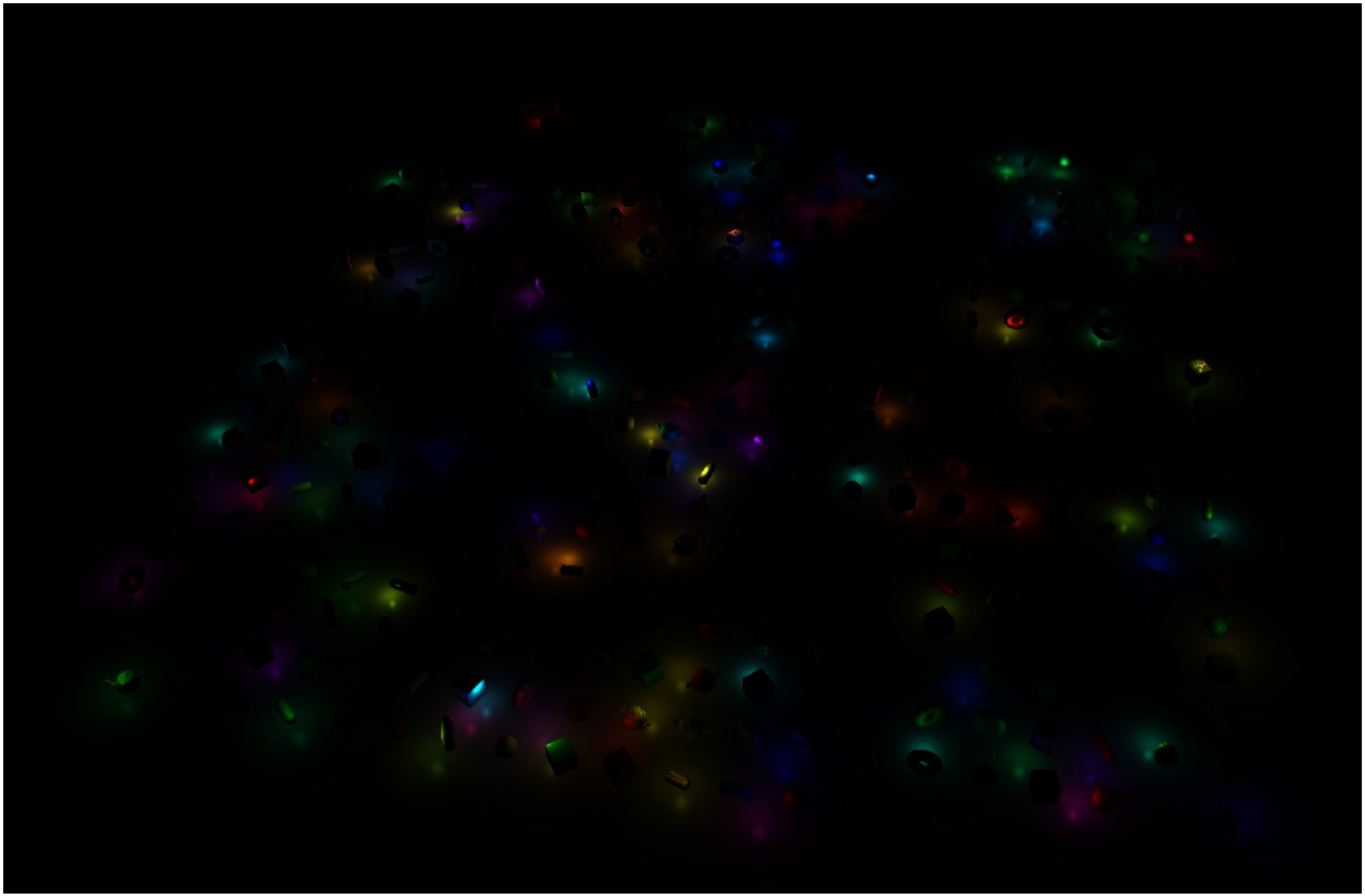
Lesson 4 – Deferred shading

PV227 – GPU Rendering

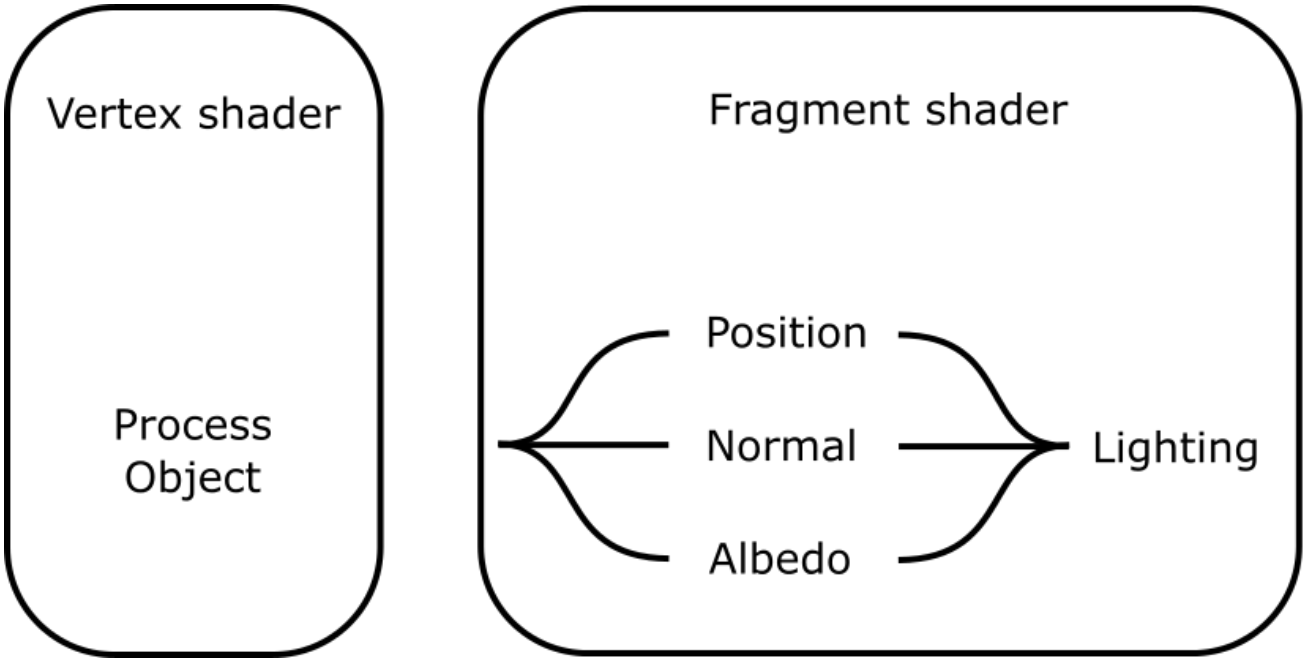
Jiří Chmelík, Jan Čejka
Fakulta informatiky Masarykovy univerzity

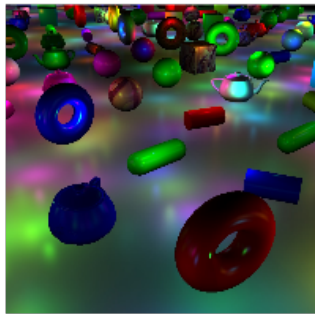
08. 10. 2019

Motivation: A loooot of lights



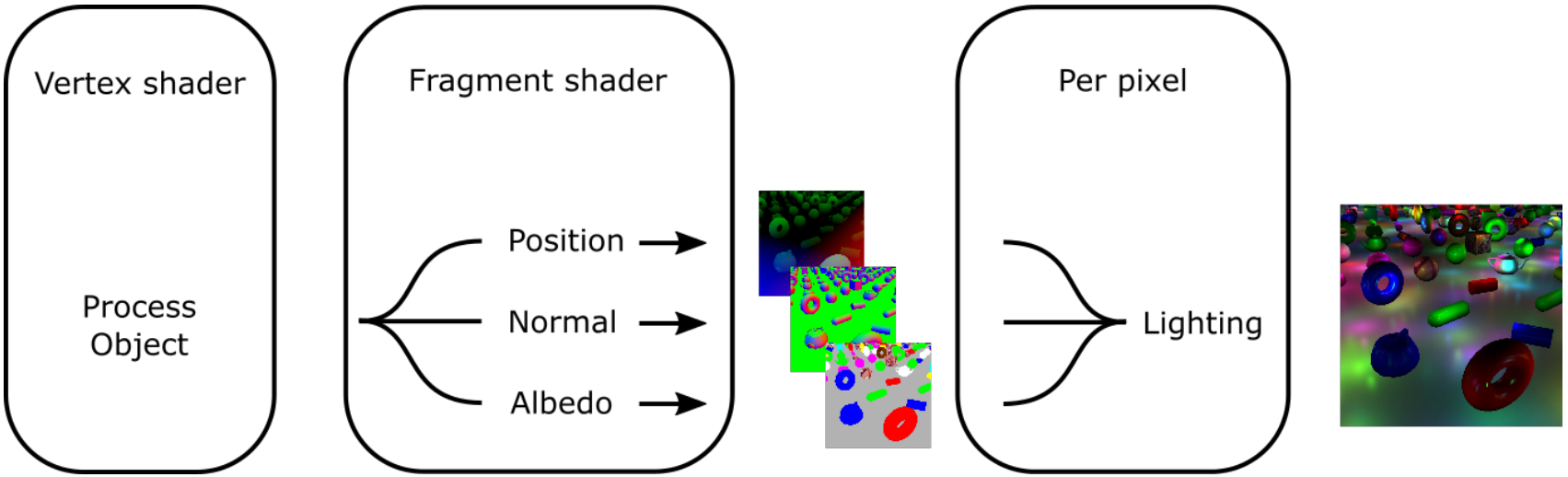
Forward and deferred shading





Forward shading

Forward and deferred shading

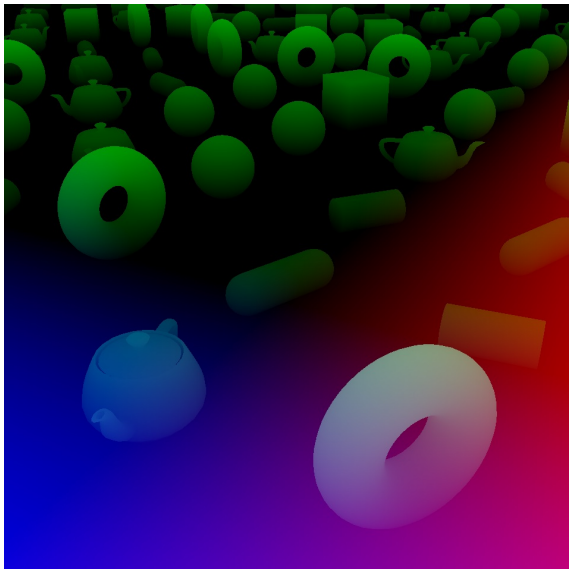


Deferred shading

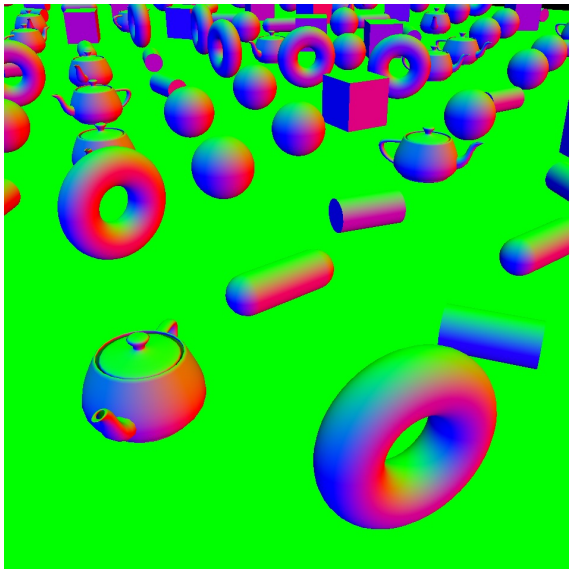
Task: Create and display G-buffer

- **Task 1**
  - Complete *notexture_deferred_fragment.glsl* and *texture_deferred_fragment.glsl*, output position, normal, and albedo.
  - Look at the result. Note that positions are scaled to 0.02 (our scene is large).

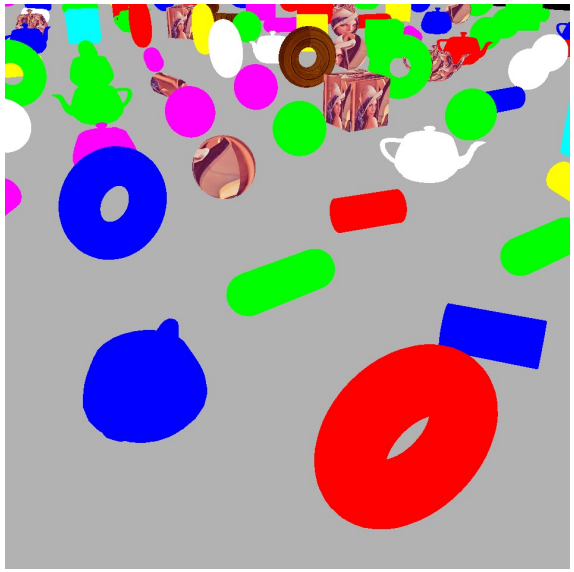
Task: Create and display G-buffer



Positions



Normals

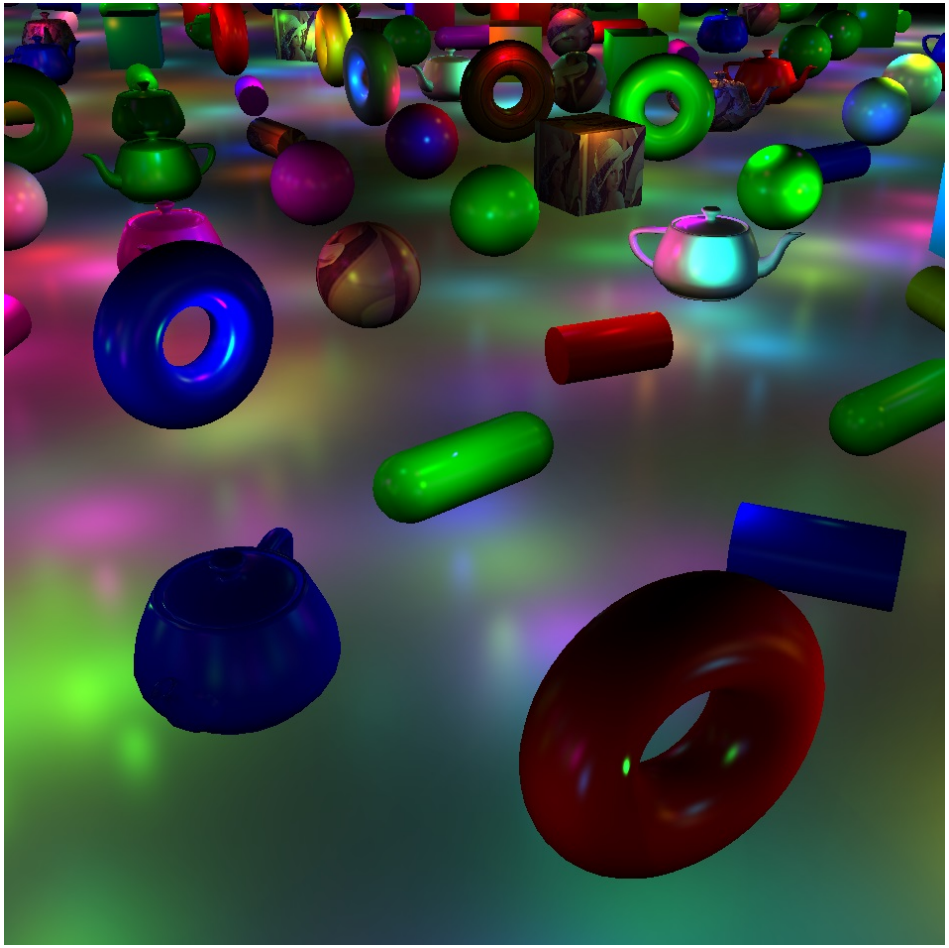


Albedo

Task: Evaluate the lighting

- **Task 2**
  - Evaluate the lighting in *evaluate_quad_fragment.glsl*.

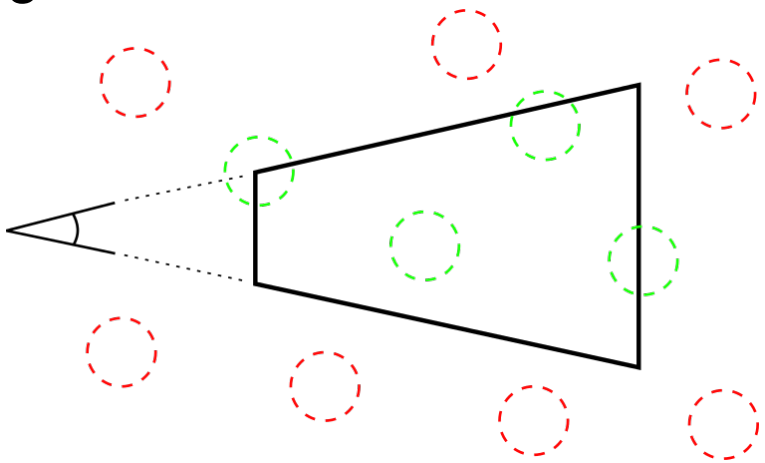
Task: Evaluate the lighting



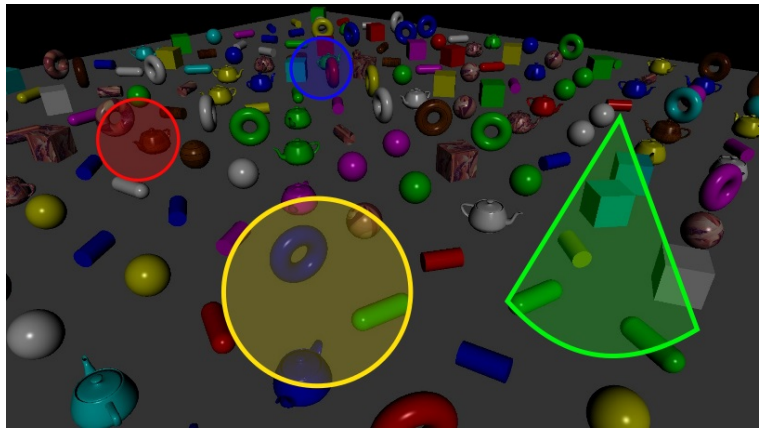
Result

Reducing the number of evaluated lights

- View-frustum culling



- Light volume



Light volume implementation

- Render one sphere per light, use instancing :-)
- In vertex shader, place the sphere at the position of the light, and scale it to the range of the light
- In fragment shader, evaluate the light
- Add all lights together using blending

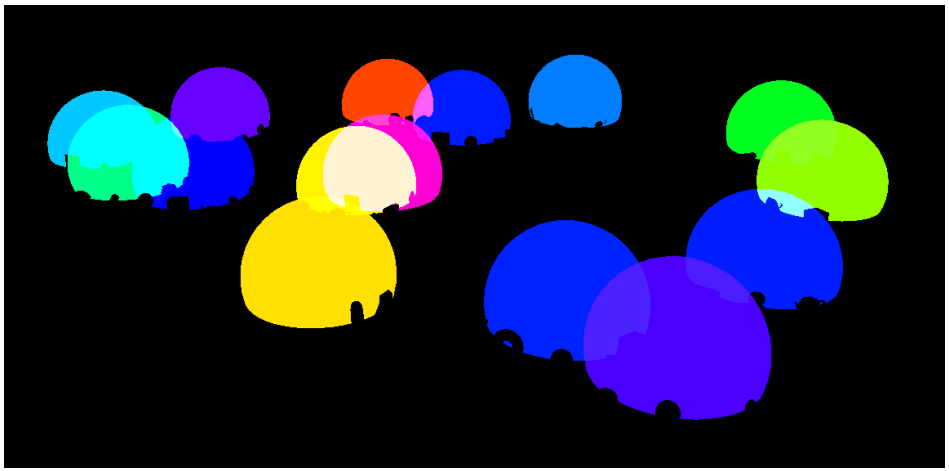
Light volume implementation

- Use depth test
  - Don’t evaluate pixels if the light is behind something.
  - We need the depth texture of the scene, in addition to the color buffer.
  - Only compare the depth with the scene, don’t render the spheres into the depth buffer.
- Render front faces only
- Use blending to sum the lighting, configure the blend function properly

Task: Implement light volume algorithm

- **Task 3**
  - Configurate the depth test, culling, and blending in C++ code
  - In *evaluate_sphere_vertex.glsl*, place the sphere to the position of the light. Also, sphere's radius is one, so enlarge it to the range of the light (*light_range* constant).
  - In *evaluate_sphere_fragment.glsl*, do not evaluate the light yet, output only the diffuse color of the light.

Task: Implement light volume algorithm



Result

Task: Implement light volume algorithm

- **Task 4**
  - Evaluate the lighting in *evaluate_sphere_fragment.glsl*.

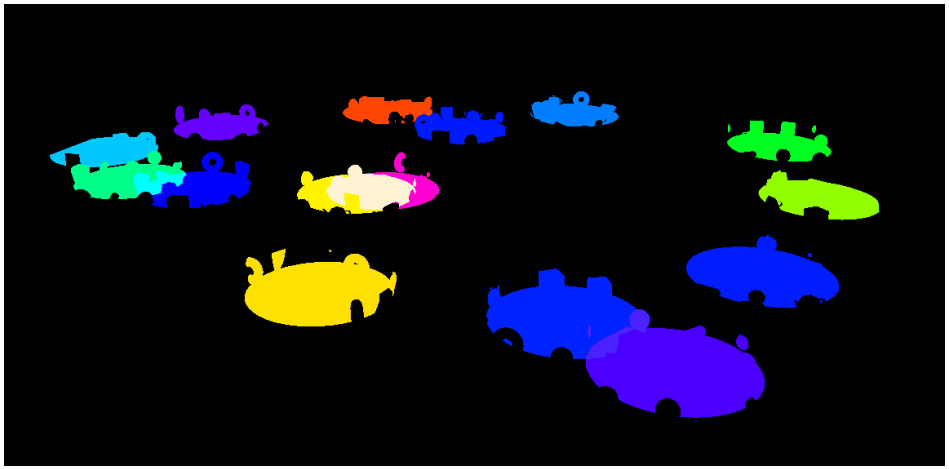
Task: Implement light volume algorithm

Result

Task: Implement light volume algorithm

- **Task 5**
  - In *evaluate_sphere_fragment.glsl*, discard fragments that are too far from the light.

Task: Implement light volume algorithm



Result (when displaying only the diffuse color of the light)

Pros and cons

Pros:

- Lighting (one of the most expensive operations) is evaluated only once per pixel.
- Lighting is processed independently, reducing the number of combinations of materials and lighting.
- G-buffer is good for other postprocessing effects.

Pros and cons

Cons:

- Does not work with multisampling.
- Transparency
- G-buffer needs a lot of memory (ours: 3x16B/pixel, 100 MB FullHD)
- Materials must not be too much complicated (the space of G-buffer is limited)

Things we used

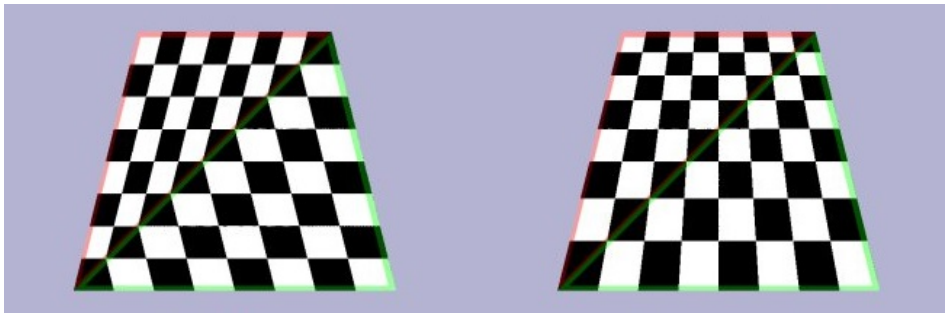
OpenGL queries, very briefly:

- *glGenQueries* generates a query object
- *glBeginQuery(GL_TIME_ELAPSED, obj)* starts measuring of the time, only one measuing may run at the same time.
- *glEndQuery(GL_TIME_ELAPSED)* stops measuring.
- GPU time is measured, not the CPU time.
- *glGetQueryObjectiv(obj, GL_QUERY_RESULT_AVAILABLE, result)* returns 1 if the result of the query is available.
- *glGetQueryObjectui64v(obj, GL_QUERY_RESULT, result)* returns the result of the query.

Things we used

Interpolation of variables between vertex shader and fragment shader

- Usage: *flat int light_idx*, in vertex and fragment shader.
- *smooth*: default interpolation, interpolates the values between vertices with perspective correction.
- *noperspective*: linear interpolation, i.e. without perspective correction.
- *flat*: no interpolation, the value of the “provoking” vertex is used. Necessary for integers.



Interpolation without and with perspective correction. Source Wikipedia.

Things we used

Shader storage buffer objects (SSBO):

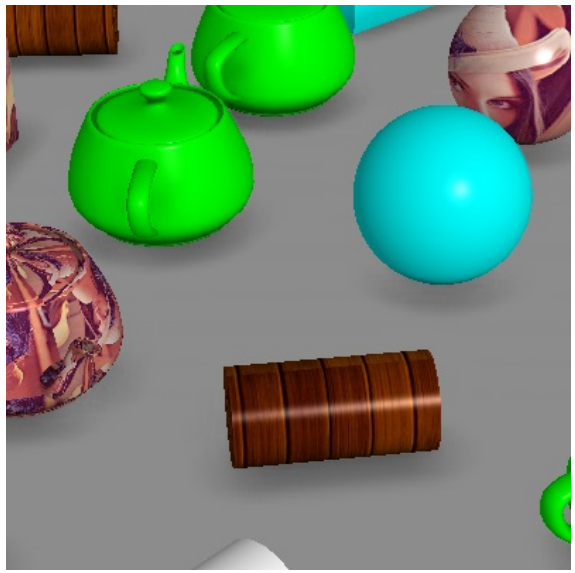
- Since OpenGL 4.3
- Similar to uniform buffers, similar usage:
  - Use *GL_SHADER_STORAGE_BUFFER* instead of GL_UNIFORM_BUFFER
  - Use *buffer* instead of *uniform* in shaders
- Much larger than uniform buffers (at least 128 MB, but usually limited by the memory of the GPU)
- Shaders can write to them (preferably using atomic operations)

Try changing *LIGHTS_MAX_COUNT* and *LIGHTS_STORAGE* in the project (6 places) to use shader storage buffers and 1000 lights. Don't use forward shading for this :-)

Next lecture

Using G-buffer for:

- Screen-space ambient occlusion
- Depth of field



Screen-space ambient occlusion